DTIC FILE COPY

(4)

RADC-TR-88-75
Final Technical Report
June 1988

AD-A197 858

RADC SYSTEM/SOFTWARE REQUIREMENTS ENGINEERING TESTBED RESEARCH & DEVELOPMENT PROGRAM

International Software Systems, Inc.

DTIC
ELECTE
AUG 1 2 1988
S D
H

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

88 8 [illegible]

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-88-75 has been reviewed and is approved for publication.

APPROVED: William E. Rzepka

WILLIAM E. RZEPKA
Project Engineer

APPROVED: Raymond P. Urtz Jr.

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: John A. Ritz

JOHN A. RITZ
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COEE) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notice on a specific document requires that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS N/A
2a SECURITY CLASSIFICATION AUTHORITY N/A	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) TR-F101-85-0129-C	5 MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-88-75
6a NAME OF PERFORMING ORGANIZATION International Software Systems, Inc. 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION Rome Air Development Center (COEE)
6c ADDRESS (City, State, and ZIP Code) 9420 Research Blvd, Suite 200 Austin TX 78759	7b ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center 8b OFFICE SYMBOL (If applicable) COEE	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-85-C-0129
8c ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 62702F; PROJECT NO: 5581; TASK NO: 22; WORK UNIT ACCESSION NO: 17
11 TITLE (Include Security Classification) RADC SYSTEM/SOFTWARE REQUIREMENTS ENGINEERING TESTBED RESEARCH & DEVELOPMENT PROGRAM	
12 PERSONAL AUTHOR(S) R. Balzer, M. Konrad, C. Ramamoorthy, W. Royce, W. Rzepka, S. Sherman, L. Stucki, T. Welch, R. Yeh	
13a TYPE OF REPORT Interim 13b TIME COVERED FROM Jul85 TO Jul86	14 DATE OF REPORT (Year, Month, Day) June 1988 15 PAGE COUNT 230
16 SUPPLEMENTARY NOTATION N/A	

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB-GROUP		
15	07		Requirements	Scenario
12	05		Requirements Analysis	Formal Specification
			Requirements Engineering Prototyping	(See Reverse)

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This report presents the results of a panel, sponsored by the Rome Air Development Center (RADC), to define a ten-year R&D program for requirements engineering.

The R&D program will assist ~~RADC~~ in the development of a Requirements Engineering Testbed of tools and methodologies for defining the requirements of planned operational systems and for examining research issues in requirements engineering.

The panel recommended an approach which included two R&D tracks.

The "Formal Language Track" attempts to formally represent requirements and specifications in the same executable language and to develop analysis tools which provide feedback on the requirements and specifications. Such analysis includes treating the specification as a prototype and determining whether the requirements are satisfied.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED	
22a NAME OF RESPONSIBLE INDIVIDUAL William E. Rzepka	22b TELEPHONE (Include Area Code) (315) 330-2762	22c OFFICE SYMBOL RADC (COEE)

DD Form 1473, JUN 86 — Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

The "Evolutionary Track" is based on extending existing requirements and specification techniques. It emphasizes prototyping with scenario generation and execution capabilities in the near/midterm, and then selectively enhancing the associated methods, techniques and tools over the long term.

In addition to this research approach, the Evolutionary Track attempts testbed integration through development of a common user interface and database for tools.

In the course of its work, the panel created a generic model of the requirements engineering process, which proved to be of value in many requirements engineering discussions. The panel also characterized the background and objectives of testbed users and analyzed the possible R&D issues in the two tracks.

18. SUBJECT TERMS (Continued).

Evaluation Testbed
Requirements Engineering Process Model

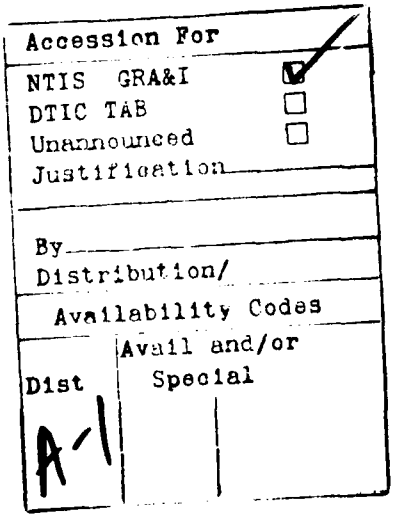


TABLE OF CONTENTS

LIST OF FIGURES iv

1.0 EXECUTIVE SUMMARY 1

1.1 GENERAL BACKGROUND AND RADC'S GOALS 1
1.2 PANEL FINDINGS AND RECOMMENDATIONS 1
1.3 OVERVIEW OF THE RESEARCH AND DEVELOPMENT PROGRAM 2

2.0 INTRODUCTION 5

2.1 AIR FORCE REQUIREMENTS PROBLEM 5
2.2 RADC'S SOLUTION TO THE AIR FORCE REQUIREMENTS PROBLEM - THE REQUIREMENTS ENGINEERING TESTBED (RET) CONCEPT 6
2.3 REQUIREMENTS PANEL OBJECTIVES 9
2.4 CURRENT STATE OF THE REQUIREMENTS PROCESS 9

3.0 THE REQUIREMENTS ENGINEERING PROCESS MODEL 13

3.1 THE MODEL 13
3.2 A WALK-THROUGH 15
3.3 SCENARIOS 17
3.3.1 1990 SCENARIO 20
3.3.2 1995 SCENARIO 36

4.0 PANEL RESEARCH AND DEVELOPMENT RECOMMENDATIONS 44

4.1 NEAR-TERM INTEGRATION OF RET 44

4.2 REQUIREMENTS ENGINEERING TESTBED (RET) RESEARCH AND DEVELOPMENT PROGRAM 45
4.2.1 TWO-TRACK PROGRAM 45
4.2.2 EVOLUTIONARY TRACK 48
4.2.2.1 OBJECTIVES AND ROADMAPS 49
4.2.2.2 RESEARCH AND DEVELOPMENT ISSUES 54
4.2.3 FORMAL LANGUAGE TRACK 61
4.2.3.1 OBJECTIVES AND ROADMAP 61
4.2.3.2 RESEARCH ISSUES 61
4.2.4 RESOURCE ALLOCATION 66
4.2.4.1 PRIORITIZATION IN EVOLUTIONARY TRACK 66
4.2.4.2 PRIORITIZATION IN FORMAL LANGUAGE TRACK 67
4.2.4.3 ALLOCATION OF RESOURCES TO BOTH TRACKS 67

5.0 CONCLUSIONS 68

6.0 EPILOGUE: RESULTS FROM A REVIEW OF THIS REPORT 70

APPENDIX A: CHARACTERIZATION OF TESTBED USERS A-1

APPENDIX B: DOD STANDARD 2167 AND
THE REQUIREMENTS ENGINEERING TESTBED PROCESS MODEL B-1

APPENDIX C: PROCESS MODEL: DETAILED CHARACTERIZATION C-1

C.1 DEFINITIONS C-1
C.2 PROCESS MODEL ACTIVITIES AND SUPPORT FROM 1990/1995 TOOLS C-5
C.3 EXPRESSION OF INFORMATION IN THE 1990 RET C-11
C.4 GENERAL OBSERVATIONS C-14

APPENDIX D: REQUIREMENTS ENGINEERING TESTBED (RET) TOOLS AND
ARCHITECTURE D-1
D.1 TOOLS CURRENTLY UNDER DEVELOPMENT D-1
D.2 LOOSE COUPLING STRATEGY D-4
D.3 INTEGRATION STRATEGY D-5
D.4 LONG-RANGE ARCHITECTURE D-8

APPENDIX E: DETAILED DESCRIPTIONS OF REQUIREMENTS ENGINEERING TESTBED
(RET) RESEARCH AND DEVELOPMENT ISSUES E-1

E.1 EVOLUTIONARY TRACK E-2

RESEARCH ISSUES:
E.1.1 GOALS AND REQUIREMENTS SYNTHESIS E-2
E.1.2 DOMAIN MODELS AND INFORMATION E-4
E.1.3 REQUIREMENTS - STATIC ANALYSIS E-10
E.1.4 REQUIREMENTS ANALYSIS METHODOLOGY E-15
E.1.5 DYNAMIC ANALYSIS E-20
E.1.6 SOLUTION ARCHITECTURE SYNTHESIS E-28
E.1.7 SCENARIO GENERATION SUPPORT
& SCENARIO COVERAGE ANALYSIS E-30
E.1.8 VALIDATION OF PROTOTYPE AND SCENARIOS E-38
E.1.9 SCENARIOS EXECUTION AND ANALYSIS OF RESULTS E-40
E.1.10 ESTIMATION OF COST, RISK, TIME
IN SYSTEM DEVELOPMENT;
PERFORMANCE & EXECUTION COSTS ANALYSIS E-46
E.1.11 REQUIREMENTS EVALUATION E-53
E.1.12 TESTBED EFFECTIVENESS E-62

DEVELOPMENT ISSUES:
E.1.13 USER INTERFACE E-67
E.1.14 DATABASE E-73
E.1.15 EVOLUTIONARY TESTBED INTEGRATION E-80

E.2 FORMAL LANGUAGE TRACK E-85

E.2.1 REQUIREMENTS INTEGRATED INTO SPEC LANGUAGE E-88
E.2.2 FORMAL INTERPRETATION OF REQUIREMENTS AGAINST BEHAVIOR E-89
E.2.3 METHODOLOGY FOR FORMAL REQUIREMENTS SYNTHESIS E-90
E.2.4 GOAL COVERAGE ANALYSIS E-91
E.2.5 MULTIPLE LEVELS OF ABSTRACTION E-92
E.2.6 SCENARIO GENERATION AND COVERAGE E-94
E.2.7 INCREMENTAL REQUIREMENTS LANGUAGE E-96
E.2.8 MANAGING RESOURCES E-98

APPENDIX F: PHILOSOPHICAL CONSIDERATIONS F-1

F.1 ASSESSING COMPLETENESS IN REQUIREMENTS F-1
F.2 UNEXPECTED BOUNDING OF THE SOLUTION SPACE F-1

APPENDIX G: REFERENCES G-1

LIST OF FIGURES

Number		Page
2.2-1	Requirements Engineering Testbed Users Characterization	8
3.1-1	Requirements Engineering Process Model	14
3.3-1	Engineering Context Description	19
3.3.1-1	PMS Viewpoints	22
3.3.1-2	Domain Model: Role characterizations	23
3.3.1-3	Domain Model: Example transaction: "Notify doctor on unsafe condition"	22
3.3.1-4	Goals	22
3.3.1-5	First Solution - Refinement of PMS Viewpoint	24
3.3.1-6	First Solution - Extended Domain Model	24
3.3.1-7	Revised Solution - Refinement of PMS Viewpoints	24
3.3.1-8	Revised Solution - Extended Domain Model	27
3.3.1-9	Revised Solution - Extended Domain Model Showing Two Patient Transactions	28
3.3.1-10	Requirements - Excerpt Showing Doctor's Goal Has Been Revised	27
3.3.1-11	Functional Prototype - Ward Station - An Excerpt	31
3.3.1-12	Interface Prototype - Nurse Display Mock-up and Comments Elicited	32
3.3.1-13	PMS Stress Scenario - Excerpt	33
3.3.1-14	Final Requirements & Solution Architecture - Only New Changes Shown	35
3.3.2-1	Domain Model	38
3.3.2-2	Goals	38
3.3.2-3	Refined Model and Goals	38
3.3.2-4	A Resource	39
3.3.2-5	Extending the Domain Model to Include the Resource	39
3.3.2-6	A Candidate Solution	39
3.3.2-7	Automatic Calculation of Solution Cost and Evaluation Against Goals and Requirements	39
3.3.2-8	Generation of a Scenario Depicting an Unsafe Patient	41
3.3.2-9	Automatic Analysis of the Result of Executing the Solution Against the Scenario	41
3.3.2-10	Requirements	43
3.3.2-11	Candidate Solution	43
3.3.2-12	Revised Requirements - Explanation of Modification	43
4.2.1-1	RET R&D Roadmap	46
4.2.2.1-1	Evolutionary Track Roadmap: R&D Efforts to Aid Making Better Analysis on Requirements	50
4.2.2.1-2	Evolutionary Track Roadmap: R&D Efforts Facilitating Creation of Prototypes & Scenarios and Analysis of Results	52

4.2.2.1-3	Evolutionary Track Roadmap: R&D Efforts Facilitating Measurement of Tools in an Integrated Testbed	53
4.2.3.1-1	Formal Language Track R&D Roadmap	62
E.1.2-1	Domain Models R&D Effort	E-9
E.1.3-1	Requirements - R&D Effort	E-14
E.1.4-1	Requirements Analysis Methodology R&D Effort	E-19
E.1.5-1	Dynamic Analysis R&D Effort	E-27
E.1.7-1	Scenario Generation Support & Scenario Coverage Analysis R&D Effort	E-37
E.1.9-1	Scenario Execution and Analysis of Results R&D Efforts	E-45
E.1.10-1	Estimation of Time, Cost, Risk in System Development, Performance & Execution Costs Analysis R&D Efforts	E-52
E.1.11-1	Requirements Evaluation R&D Effort	E-61
E.1.12-1	Testbed Effectiveness R&D Effort	E-66
E.1.13-1	User Interface R&D Effort	E-72
E.1.14-1	Database R&D Effort	E-79
E.1.15-1	Evolutionary Testbed Integration R&D Effort	E-84
E.2-1	Formal Language Approach Roadmap	E-87

1.0 EXECUTIVE SUMMARY

A panel of computer scientists was tasked to define a ten-year research and development program for RADC's Requirements Engineering Testbed. The purpose of this report is to present the panel's findings and recommendations.

1.1 General Background and RADC's Goals

RADC's charter to the panel, to define an R&D program, was based on existing RADC goals and plans. These are discussed below.

Incorrect requirements lead to the development of systems that cannot be fielded because they do not address all critical mission needs. Within the Air Force, existing requirements tools and methods go largely unused because they fall short of what is needed to produce correct requirements for complex systems and software. Even among software contractors, existing tools and methods have not been quickly adopted. Air Force users have great difficulty in recognizing and tracking their mission concerns in requirements specifications, because they tend to be large, complex, and written for the technical community. Thus, Air Force users need better tools and methods.

Therefore, RADC proposes to develop a Requirements Engineering Testbed (RET) to support the development and evaluation of new tools and methods. Air Force users would use the RET to define and exercise their requirements for planned operational systems.

A secondary goal of RADC is to encourage use of the new tools by Air Force users and industry; the RET would be the vehicle.

In the near term, RADC is developing prototyping tools that will assist Air Force users in requirements understanding and in investigating their implications. RADC plans to have these tools hosted in the RET by 1988, with additional requirements engineering capabilities added by 1990.

1.2 Panel Findings and Recommendations

To help achieve their RET goals, RADC identified the need for a ten-year research and development (R&D) program to guide them in the development of new tools and techniques. RADC appointed a panel of computer scientists from academia and industry to define such a program.

To help define the objectives for the R&D program, the panel created:

(1) a model of the requirements engineering process, capturing the panel's understanding of that process; and

(2) two scenarios illustrating the process model in terms of RET capabilities, one for 1990 and the other for 1995.

The model provided a basis for defining the R&D program: the R&D program would have to provide tools and methods to support process model activities. The scenarios provided a complementary basis: the R&D program would have to provide tools and methods that would work together in the way envisioned and provide the illustrated capabilities.

The panel then defined the R&D program as the strategy to bring all this about.

Recommendations

The panel recommended that RADC pursue:

A research and development program for the Requirements Engineering Testbed (RET) consisting of two tracks: (1) an Evolutionary Track for developing tools and methods such as rapid prototyping that in the near term give the best payoff in better requirements, and (2) a Formal Language Track for exploring the higher risk/payoff implications of a formal requirements language. The risk in the Formal Language Track is that one must be able to express requirements formally. The payoff is in the formal activities that can be automated. Determining requirements satisfaction and generation of scenarios are examples.

The panel's 1990 and 1995 RET scenarios respectively illustrate capabilities that would be made available through the Evolutionary and Formal Language Tracks. The panel concluded that both tracks should provide the RET user with better analysis and trade-off capabilities than are currently available.

The panel further recommended a testbed integration plan for 1990, that would lead to a uniform user interface to all tools and a common repository for all requirements, designs, and tool data, and which incorporates RADC's currently-contracted requirements tools. This recommendation is based on a short-term goal of a loose coupling of the tools.

Report Review

A review of this report was conducted. The review results are presented in an epilogue (section 6.0). In general, the report was found to be technically sound.

1.3 Overview of the Research and Development Program

The R&D program consists of R&D thrusts in these areas: (1) prototyping, (2) requirements analysis, (3) tool integration and evaluation, and (4) a formal language for requirements and specifications. The objective of the Evolutionary Track is to provide better capabilities and more automation in the first two areas and to accomplish and support the third area. The focus of the Formal Language Track is the last area. Below, we summarize

each area, identifying capabilities to be developed by 1990 and 1995, and how these relate to RET goals.

Prototyping

In prototyping, the 1990 goal is to develop capabilities in: (1) prototyping system interfaces and functions, (2) developing scenarios to drive the prototypes in experiments, and (3) collecting and analyzing results. The 1995 goal is to extend capabilities for analyzing prototyping results.

Relationship to RET goals: End users are generally able to analyze their mission concerns better in operational settings than by reading lengthy, textual requirements specifications. Thus prototyping will play an important role in discovering and understanding critical needs, and therefore in capturing these needs in the requirements. Prototyping will also provide a feasibility check, and when later supplemented with performance analysis, will serve as a basis for sensitivity analysis on requirements.

Requirements Analysis

In requirements analysis, the 1990-1995 goals are to be able to analyze requirements to investigate assumptions, decisions, implications, and requirements quality: (1) expected cost and risk of developing the envisioned system, (2) expected cost, performance, and reliability of operating the envisioned system, (3) the rationale for a particular mission need or trade-off decision, (4) requirements related to specific needs and expectations of a particular end user, and (5) which requirements are untestable, contradictory, or redundant. The key to achieving these goals will be to represent requirements more precisely, strongly couple the analysis to requirements updates, and identify relevant metrics. Another goal is to provide a methodology to guide the user in the use of these analysis capabilities.

Relationship to RET goals: Requirements correctness and quality will improve. Air Force users will use the analysis tools to investigate their concerns in the requirements including the rationale and implications of decisions that were made.

Tool Integration and Evaluation

In the integration and evaluation of tools, the 1990 goal is to provide state-of-the-art database and human interface capabilities as a basis for an integrated testbed of tools and for monitoring tool performance. The 1995 goal is to integrate new prototyping, requirements analysis, and formal language capabilities into the instrumented RET to facilitate their evaluation.

Relationship to RET goals: A common database implies tools will be able to share data. A common user interface implies user learning time will be reduced. Both imply the user will be able

to move easily from one tool to another. Development efforts will be reduced - new tools can use the same data management facilities and utilize the same human interface mechanisms for user communication. An instrumented and integrated testbed is a basis for evaluating relative tool effectiveness.

Formal Language

In the formal language area, the 1990 goal is to provide: (1) a common formal language for requirements and specifications and (2) tools that automatically compare requirements and specifications statements. The 1995 goal is to provide: (1) the capability to generate and interpret scenarios and (2) extensions to the language that facilitate incremental modifications and multiple levels of abstraction.

Relationship to RET goals: Requirements statements will be precise and machine-interpretable, leading to increased automation and support for prototyping, requirements analysis, and development activities. These implications are long-term consequences of the formal language approach.

2.0 INTRODUCTION

The purpose of this section is to establish a context for understanding the panel's recommendations. The panel's recommendations are presented in sections 3 and 4.

In summary, the context is this: (1) the Air Force needs better requirements tools and techniques, (2) RADC's approach to meeting this need is to create a testbed for development and evaluation of new requirements tools and techniques, (3) RADC convenes a panel of computer scientists to refine the testbed plan and to recommend a testbed research and development program, and (4) the panel makes recommendations based on the current state of the requirements process: available tools, characteristics of requirements, and methods of checking requirements.

Before dealing with the context in detail, we define some terms.

Requirements Terminology

Requirements are precise statements of need which characterize a needed system in terms of its external characteristics (especially interfaces and functionality visible to the user) and constraints (e.g. on execution performance and reliability, and on development cost and time).

Requirements engineering is the systematic application of tools and techniques to guide and control the emergence of the requirements product. It is an iterative process of analysis, evaluation, and refinement. Its inputs are mission-related ideas and problems expressed by mission specialists and their representatives. Its output is the set of requirements.

2.1 Air Force Requirements Problem

As part of an RADC-sponsored effort, an informal survey of Air Force users (specifically, mission users and acquisition engineers) and contractors was conducted. The survey consisted of 25 interviews with both Air Force users, other Department of Defense users, and contractors. The survey identified three major problem areas:

> (1) Requirements specifications were written for procurers (acquisition engineers) and their technical staffs. Thus mission users found them to be too technical and felt "shut out".
>
> (2) Mission users and contractors found it difficult to relate A-spec to B-spec (i.e. high-level system specification to software requirements) because of the significant "gulf" between them.

(3) Contractors and mission users complained that traceability could only be demonstrated manually, making it hard to assess requirements coverage.

In summary, Air Force users found it difficult to preserve a recognizable representation of mission user concerns in the derived specifications.

The survey also found that most mission users who could have used a requirements method/tool (e.g. PSL/PSA) never did, and not one of those who did considered the use to be a success. So requirements engineering methods and tools are underutilized by Air Force users. They are based on technologies such as: formal specification languages, consistency checkers, and report generators. Those methods and tools that are not difficult to master do little more than check how well things fit together syntactically. Instead of stating his needs and making decisions at the mission level, the user is forced to work at a design level or use the English language. The conclusion: existing methods and tools match the needs and abilities of system developers instead of Air Force users.

2.2 RADC'S Solution to the Air Force Requirements Problem - the Requirements Engineering Testbed (RET) Concept

The Requirements Engineering Testbed

To deal with the Air Force requirements problem, RADC's long-term goal is to develop and evaluate new tools and methods that will enable Air Force users to bring new technologies to bear on their requirements engineering problems.

But these new tools and methods cannot be developed in isolation. They need to be evaluated on realistic problems and placed into an environment that supports their use by Air Force users.

Therefore RADC proposes the creation of a facility which promotes experimentation, called the Requirements Engineering Testbed (RET), and which hosts the new methods and tools. Air Force users will use the RET to define, analyze, and exercise requirements of planned operational systems. This will have two benefits. First, new tools and methods will have early exposure on realistic problems, encouraging their use by Air Force users and industry. Second, evaluation of the new tools and methods in terms of requirements quality and productivity will provide insight for the next generation of tools.

We next characterize the users of the RET. This is followed by a discussion on hosting RADC's currently-contracted prototyping tools in the RET in early 1988, and a subsequent integration.

RET Users

The RET must support the requirements engineering activities of three user classes in the evaluation of new tools and methods: Air Force mission users, Air Force acquisition engineers, and contractor software developers. Figure 2.2-1 defines their jobs and characterizes their present and projected future familiarity with computers and programming.

The RET must concentrate on supporting the mission user and acquisition engineer in keeping with RADC's mission and because support for these classes has been neglected by the current generation of tools.

These two users have different requirements concerns and interests. Mission users are interested in: the data (messages) which must be processed, the user interface, performance (e.g. workstation ergonomics and work flow), and scenarios. Acquisition engineers are interested in: functional characteristics of the target system, its software requirements (database management system, utilities), testing, and performance.

In order to better understand the context of the RET, the panel developed a more detailed characterization of the three user classes. Appendix A presents the panel's characterization.

Near-Term RET Prototyping Tools.

RADC currently has three different prototyping tools under contract to be developed. In the near term (early 1988), these three tools will be hosted by the RET and will provide its initial requirements capabilities:

(1) The Analyst, an existing requirements tool currently being enhanced by Systems Designers under subcontract to Imperial College, will assist in the creation of a semiformal problem domain description. The tool also aids documentation of performance and reliability stress points. A prototyping capability will estimate performance based on estimated work loads.

(2) The VHLL Prototyping tools, to be developed by International Software Systems, will assist in the creation of functional prototypes. A VHLL (very high level language) is used to specify the desired functionality of the target system. The resulting description is executable and can be exercised with different inputs dynamically.

(3) The Rapid Prototyping System, to be developed by Martin Marietta, takes a model-based approach to prototyping; aiding the production of operator models, processor models, scenario-driven functional models, and communications models. Within these models, system performance can be estimated based on estimated work loads and target system

USER CLASS:	MISSION USER	ACQUISITION ENGINEER	SOFTWARE DEVELOPER
JOB:	Perform the mission Describe needs	Analyze, document, and validate requirements Procures system	Elaborate/refine requirements Develop system architecture
COMPUTER FAMILIARITY	Computer literate	Shared dumb terminal & some PCs	Local Area Network (LAN) access
PRESENT:	Text oriented, time shared terminals	Word processor/spreadsheet	Programming tools
5 YEARS:	PC per desk, graphics interface Informal methods, minimal training	PC per desk Limited prototyping	Multiple languages/workstations Integrated development environment Loosely coupled tools
10 YEARS:	Network Mission language	LAN access to several computers Prototyping	Integrated development environment, tightly coupled tools Prototyping testbed

FIGURE 2.2-1 REQUIREMENTS ENGINEERING TESTBED USERS CHARACTERIZATION.

configuration. There is also a capability provided for rapidly designing interfaces with dynamic graphics.

RADC plans an integration effort, which would start soon after delivery of these tools, and would result in a 1990 testbed of integrated tools.

2.3 Requirements Panel Objectives

To properly plan the introduction of new methods and tools into the RET and to help in their evaluation, a long-term research and development (R&D) program needed to be defined. To assist in the definition of the R&D program, a panel of computer scientists from academia and industry was appointed and convened four times during July 1985 to February 1986.

The members of the panel were:

Robert Balzer - Information Sci. Inst., Marina del Rey, CA
* Michael Konrad - International Software Systems, Austin, TX
C. V. Ramamoorthy - University of California at Berkeley, CA
Winston Royce - Lockheed Missiles & Space, Austin, TX
* William Rzepka - Rome Air Development Center, Rome, NY
Steve Sherman - Lockheed Missiles & Space, Austin, TX
Leon Stucki - Future Tech, Auburn, WA
* Terry Welch - International Software Systems, Austin, TX
Raymond Yeh - International Software Systems, Austin, TX

---

* Rzepka was panel sponsor. Welch was panel chairperson. Konrad was panel report editor.

Pei Hsia of the University of Texas at Arlington, Texas, contributed to the panel's efforts.

As stated earlier, the primary objective of the panel was to define a ten-year R&D program.

A secondary panel objective was to identify what additional effort was required to make a full range of requirements engineering capabilities accessible to the RET user by 1990, based on the currently contracted tools.

The remainder of this report presents the panel's findings and recommendations, except section 6.0, which presents results of a review of the panel's findings and recommendations.

2.4 Current State of the Requirements Process

In this section we characterize the current state of the requirements process as viewed by the panel.

The current state of the requirements process is that it is almost entirely a manual process whose success depends on the insight of the analyst doing the work. There are no generally accepted methodologies, criteria for quality, or notations.

Requirements are almost entirely English text, so they are subject to differing interpretations by mission experts and by system developers, namely the two audiences that the requirements statements should bring together.

Evaluation of Existing Requirements Techniques

Existing requirements techniques, such as SREM and PSL/PSA, provide a methodology for requirements analysis. The methodology is supported by tools: (1) machine-processable languages for stating the requirements, (2) static analysis tools for determining consistency and completeness of the statements, (3) simulation tools for assessing feasibility, and (4) report generators. As an example we consider SREM (discussion is based on [Stone] and [Scheffer]):

(1) Requirements Statement Language. Many system characteristics can be expressed, e.g. the system interface in terms of message flow and content, and system software functions in terms of inputs, outputs, and processing dependencies. Certain characteristics are very difficult to express in the language and so analysts must rely on textual representations: real-time and near-real-time characteristics, parallel and distributed processing, man-machine interfacing.

(2) Analysis tools. Analysis is performed in batch and results are presented in generated reports. Two kinds of analysis are performed: (1) data analysis consisting of syntactic checks on language statements (i.e. detection of syntactically missing or misused elements), and (2) data flow analysis consisting of checks that consuming elements process exactly those elements named in their input. The SREM evaluators found that most requirements errors (80% or higher) are caught by the analyst while using the methodology, not by the analysis tools.

(3) Simulation tools. Simulations are partly "generated", but analysis of results is manual. The simulations aid verification of (1) system interfaces and processing relationships, and (2) analytic feasibility (is there a testable design that satisfies the accuracy requirements). In practice, too much labor is involved to justify more than verification of system interfaces.

(4) Role in system development. Role is late in the requirements phase and early in the design phase. Utility of SREM lies in defining, correcting, and analyzing the software-aspects of a system, i.e. after allocation of

requirements to functional software components. SREM provides little support in the "conceptual phase": expressing needs, assessing feasibility, trade-off studies.

Recently [Alford 1985], SREM is being extended to address concerns that arise earlier in the system requirements phase and also detailed issues that arise in later design and testing phases.

In summary, existing techniques provide a discipline for defining requirements and analyzing them, but are best at addressing late "requirements phase" concerns. They require a largely manual execution of the requirements process. It is not yet evident that the labor involved in their use is justified by the improved results obtained.

Problems in Communicating and Analyzing Requirements

The panel members shared these basic assumptions about requirements which guided their thinking:

1. Formal Descriptions. Requirements must be stated with increased precision. A formal basis for requirements will lead to machine interpretation, an important step in achieving increased productivity in the design and verification of software systems.

2. Domain Information. Requirements are often ambiguous with respect to assumptions and decisions based on the "common sense" of such things as the laws of physics, psychology of people, politics of organizations, software algorithms, application-specific technology (e.g. how does a radar work), etc. Mechanized interpretation of requirements necessitates capturing and interpreting this context information.

3. Prototyping. Communication of requirements statements to mission experts often requires building a functional prototype to animate those requirements. This is because of the poor communication capability of requirements statements and because many individuals are better able to analyze their mission needs when they see operational results.

4. Development Facilities. Requirements development, like any other manual design activity, should be supported by a workstation environment which provides rapid interaction via graphics tightly coupled to an object-oriented database. This type of facility will be commonplace in five years and provides important capabilities for humans dealing with complexity in large system designs.

Current Approaches to Checking Requirements

Below we describe four methods in use today to assess the validity of a set of requirements. Each of these methods is presently very labor intensive, but might be partially mechanized if requirements statements were more machine-manipulatable.

1. Consistency Checks. Requirements statements as a set can be checked for contradictions and completeness. Current techniques, such as PSL/PSA and SREM, provide modest forms (e.g. dataflow) of checking. Otherwise, the checks are manual.

2. Scenarios. Scenarios developed to illustrate typical or stressful system usage serve to provide a cross check on requirements. The development of precise scenarios is very time consuming, however, and checking them against requirements statements is not easy.

3. Prototyping. The development of a prototype provides a check on the implementability of the requirements, as well as providing a means of eliciting user feedback. To minimize prototype development effort, one generally restricts the requirements subset tested (e.g. omitting error handling and performance) and reduces prototype development "overhead" (e.g. documentation).

4. Cost Estimates. Major factors influencing any system design are the costs and risks in development and operation. Thus these factors must be considered when evaluating requirements. Present mechanisms for estimating cost and risk are notably intuitive and often ineffective.

Each of the above four areas warrants significant research on possible automation and/or mechanical support for human execution. Success in this work, however, certainly depends on increased precision and machine interpretation of requirements, so work on formalizing requirements descriptions is a cornerstone of requirements research.

3.0 THE REQUIREMENTS ENGINEERING PROCESS MODEL.

In an effort to come to a common understanding of the requirements engineering process, especially terminology, and to partially capture that understanding, the panel created a model of the process. This section presents that model and illustrates it with scenarios indicating RET capabilities in 1990 and 1995.

The appendices contain more discussion and further details on the process model. Appendix B relates process model terminology to the current Air Force requirements engineering practice. Appendix C gives detailed characterizations of process model information types and activities. Appendix F discusses philosophical problems inherent in requirements engineering that can have no solution in terms of a "better tool".

3.1 The Model

The model is portrayed in figure 3.1-1. The figure shows information depicted as boxes; and activities depicted as circles, ovals, and rounded-corner boxes.

The model indicates the dependencies and sequencing between activities, but is not meant to favor a particular methodology.

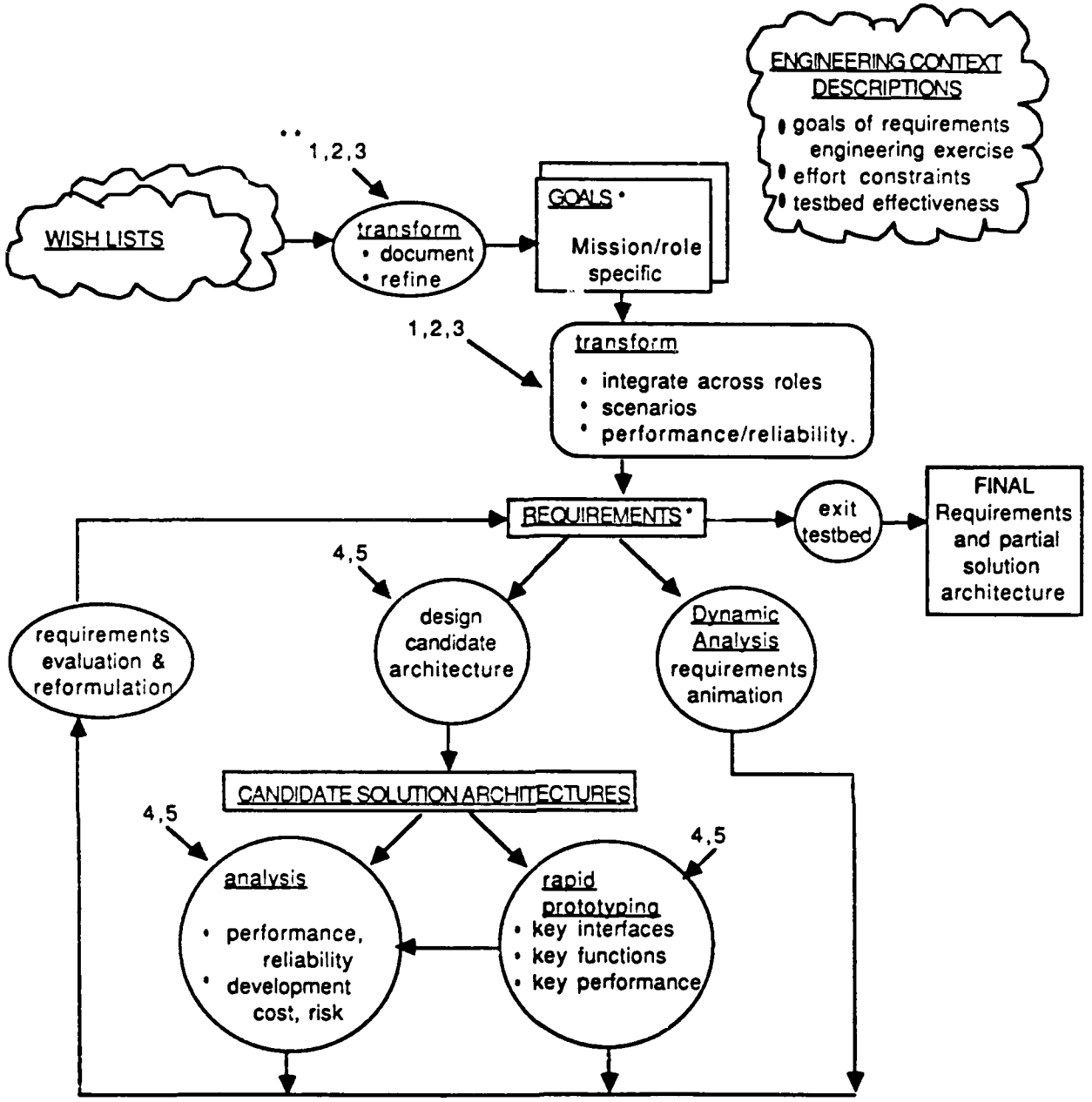
Information Types

The model identifies three major types of information: goals, requirements, and solution architectures.

Goals are expressions of objectives and needs, generally mission-related, and not necessarily feasible or consistent with each other. Mission users are the primary source.

Requirements are a consistent subset of the goals which can be feasibly realized within the available resources (especially time, money, and expertise).

A Solution architecture is a model of the target system as a composition of parts that satisfy the requirements. The more common term is specification, but the panel preferred solution architecture, as specification has been used to mean different things.



LEGEND:

**all numbered arcs reference :

DOMAIN MODELS

1. Mission Models
2. Application Domain Models
3. User cognitive Models
4. Software Resources, Algorithms
5. Hardware Models (Processors, Networks)

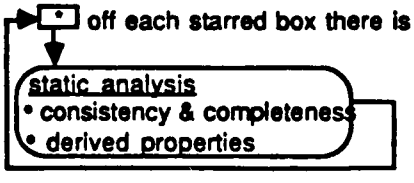


Figure 3.1-1 Requirements Engineering Process Model.

3.2 A Walk-through

We take a top-down walk through the requirements engineering process model, identifying the objects and activities depicted in the figure. For the walk-through, we assume a requirements engineer is required to produce a set of requirements for a system called the "target system".

The target system must support the different roles of its users and administrators. Thus there will be different expectations, or "viewpoints" of what the system should do, of how well it should be done, and within what cost. In the model these undocumented expectations of target system functions, performance, and cost are represented by Wish Lists. There is one wish list per role or viewpoint.

The requirements engineer is limited in the time he can expend in the creation of target system requirements. Thus he needs to prioritize his objectives. These limitations and objectives should both be documented. In the model they are represented by Engineering Context Descriptions.

Through interviews with target system users/administrators and through references to documentation of similar, existing systems and their environments, the requirements engineer collects and organizes information on the operational context of the target system. The resulting information forms a Domain Model of the environment of the target system, providing the terminology and context through which wishes can then be expressed, forming Goals. Goals represent the initial attempt at documenting a system's desired attainments. For each viewpoint, there will be one set of goals, and they should be consistent and complete within that viewpoint.

Goals are often inconsistent across viewpoints or clearly infeasible. Such difficulties must be resolved by the requirements engineer through further user interviews. The revised goals are then merged into a preliminary set of Requirements for the target system.

Through his interviews with users, the requirements engineer identifies and documents Scenarios that illustrate typical target system behavior and/or desired responses to stressful input. Scenario construction and analysis may aid stating the nonfunctional requirements, in particular, performance and reliability.

During the creation of goals and requirements, their consistency and completeness is checked. This is called Static Analysis in the process model. To determine requirements coverage, the requirements engineer might perform a walk-through, analyzing the dataflows and/or stimuli/responses through the various viewpoints. This is called Dynamic Analysis in the process model.

At this point, the requirements engineer might construct a Solution Architecture to gain better insight into: target system interfaces, functions, performance and reliability, and implied development cost and risk.

The requirements engineer creates a solution architecture by specifying how the target system is composed of parts (e.g. objects, functions) and how those parts use resources (e.g. people, software, hardware). To aid specification of resources, the requirements engineer can make reference to existing resource models.

From the solution architecture, the requirements engineer can specify a prototype. He executes the prototype against canned or user-controlled scenarios, eliciting user comments on what should be changed. All of these activities are covered by the Rapid Prototyping bubble in the process model.

Also, the requirements engineer can do Analysis directly on the solution architecture. By combining both rapid prototyping and analysis activities and iterating, the requirements engineer can do sensitivity analyses.

As a result of the insights gained through analysis and rapid prototyping, the requirements engineer determines what revisions should be made and makes them. This activity is called Requirements Evaluation & Reformulation in the process model.

There may be several iterations of prototype, analyze, evaluate and reformulate. The resulting requirements and solution architecture is called the Final Requirements and Partial Solution Architecture in the process model.

3.3 Scenarios

For the purpose of explaining the panel's vision of near-term and long-term RET capabilities, the panel developed two scenarios, one for 1990 and one for 1995. Thus these scenarios identify "where to go"; the RET R&D program, detailed in section 4, illustrates the panel's strategy of how to get there.

We make free use, below, of process model terminology and references to currently-contracted tools.

Scenario Context: 1990/1995 Objectives for the RET

The panel recommends that a primary 1990 RET objective be to provide the RET user with a wide range of requirements engineering capabilities based on three integrated tools: the Analyst, the Rapid Prototyping System, and the VHLL System Prototyping tools.

This objective is the context for the 1990 scenario, whose purpose is to illustrate these Evolutionary Track capabilities. Capabilities from the other track will not yet be generally available.

Similarly, a primary objective for the 1995 RET is to provide the RET user with a wide range of capabilities based on a single formal language for expression of requirements, solution architectures, and goals. This objective is the context for the 1995 scenario, which thus emphasizes capabilities developed under the Formal Language Track.

Scenario Starting Point

Both scenarios consider the same nontrivial application, a patient monitoring system (PMS), and address essentially the same trade-off problem: defining requirements for a PMS of low operational cost that meets the need for immediate nurse or doctor notification of a patient problem. Thus both scenarios can be compared to contrast what we envision can be done in 1990 and 1995.

The two scenarios have the same starting point: the requirements engineer visits the RET with the intent of using RET tools and techniques to define PMS requirements. As the RET is a facility for tool effectiveness experiments, careful tracking of RET user objectives and RET resources (e.g. tools and RET time) is necessary. Toward this end, the "Engineering Context Description" documents the objectives of the RET exercise and how much effort can be spent. For both 1990 and 1995, it is expected to be maintained as mostly unstructured text in the RET database.

Figure 3.3-1 illustrates the Engineering Context Description considered as context to both the 1990 and 1995 scenarios. In both scenarios, the objective of the RET exercise is to gain insight into the trade-off between two competing wishes. For the 1990 scenario, a third wish was needed to motivate the illustration of the currently contracted tools. It must be ignored when considering the 1995 scenario.

Where do these wishes come from? In a typical PMS, patients are monitored and if anything goes wrong a nurse or doctor is alerted. So the three obvious PMS users are the doctor, the nurse, and the patient. Less obvious is the hospital administrator who must show a profit. In both scenarios, it is assumed that interviews and relevant documentation have revealed these needs: (1) doctors need immediate notification if there is a serious problem (wish 1), (2) hospital administrator needs to keep operational costs down (wish 2), and (3) (only for the 1990 scenario) nurses need help tracking patients' care. Typically, investigation will reveal many needs forming "Wish Lists", but we will only deal with these three.

SYSTEM IS: PATIENT MONITORING SYSTEM

GOAL OF RET EXERCISE:

OBTAIN INSIGHT INTO THE TRADE-OFF BETWEEN THESE WISHES:

(1) DOCTOR: RAPID RELIABLE NOTIFICATION ON PATIENT PROBLEM

(2) HOSPITAL ADMINISTRATOR: LOW OPERATIONAL COST.

(3) NURSE: HELP TRACK PATIENT'S CARE.

RET EFFORT: CANNOT EXCEED 5% OF EXPECTED SYSTEM COST

1 MONTH TIME FOR DEMONSTRATION OF WHAT IS FEASIBLE

FIGURE 3.3-1 ENGINEERING CONTEXT DESCRIPTION

3.3.1 1990 Scenario

The 1990 RET will facilitate:

Partially formal characterizations of:
- The problem domain
- Functional requirements and some nonfunctional requirements
- Displays, interface protocols
- Scenarios
- Solution architectures
- Data structures (*)

Support for:
- Organizing the problem domain
- A functional description of the system
- Building executable models
- Building performance models and scenarios
- Exercising prototypes against scenarios (*)

The scenario illustrates this RET usage paradigm:

Establish RET exercise objectives
Create a "domain model" documenting user roles and activities
- This domain model will expand to include solution resources

Express goals
- Goals are expressed as attributes or annotations to the domain model

Evaluate for feasibility and trade-offs:
- Explore different solution approaches and then revise

Incorporate analyses and prototyping results (*)
Document results of RET exercise
Traceability of goals to requirements to solutions (*)

* (not illustrated in 1990 scenario)

The figures for this scenario illustrate notations that are logically equivalent to the graphics/text notations that will be available in 1990 RET tools.

Recalling figure 3.3-1, we begin the 1990 scenario. We refer to the hypothetical RET user as a "requirements engineer".

Deriving Goals

Applying the CORE method, which is supported by the Analyst tool, the requirements engineer first identifies the major PMS viewpoints (i.e. the functional roles with which PMS interacts). They might be illustrated graphically as shown in figure 3.3.1-1. The requirements engineer will interview viewpoint representatives (e.g. the head nurse), documenting all relevant interactions and wishes.

From such interviews, the requirements engineer constructs a domain model of the operational environment of the target system. Figure 3.3.1-2 illustrates a portion of such a model (underlined words are keywords). The domain model would also represent typical "transactions", e.g. "notifying doctor on unsafe condition", as illustrated in figure 3.3.1-3.

Figure 3.3.1-2 is not complete, among the activities not shown here but referenced later: the patient can request a nurse (activate a nurse alarm) and PMS will pass this patient request to a nurse.

The domain model (figures 3.3.1-2 - 3.3.1-3) provides much context necessary for expressing and interpreting Goals. Wishes constraining operational parameters such as time, reliability, and operational cost will often be expressed in 1990 as formal attributes or informal annotations to transactions (whether formal or informal is not yet clear). Two of our figure 3.3-1 wishes can be so expressed: the doctor's performance and reliability wish and the hospital administrator's cost wish. The nurse's wish will likely be expressed in informal text. See figure 3.3.1-4.

The requirement engineer's objective is to analyze the trade-off between these performance and cost wishes. To do this, the requirements engineer will treat the goals as preliminary requirements and explore solutions.

At this point in our scenario, we assume the requirements engineer interrupts application of the CORE method to better understand the solution implications. The knowledge accrued will help him refine and revise the goals into requirements.

Deriving a Candidate Solution Architecture, i.e. Solution

The requirements engineer needs to consider how to build a system that will satisfy the Requirements. He decides on the following approach:

(1) For monitoring vital signs and detecting unsafeness, use a "hardware monitor" (one per patient).

(2) For doctor notification, locate a "ward station" in each ward. The nurse(s) will have responsibility for patients in his/her ward and will notify the doctor on any unsafe patient condition as reported by the ward station.

Figure 3.3.1-5 shows PMS functionality allocated to hardware monitors and ward stations. Figure 3.3.1-6 gives partial characterizations of a hardware monitor, a ward station, and how they constitute a PMS solution. The notation employed is the same as in figure 3.3.1-2. A (probably manual) check will verify that the composition of their activities is a correct refinement of PMS activities of figure 3.3.1-2.

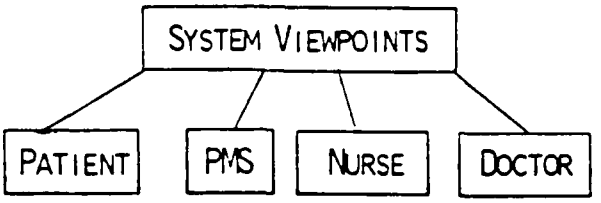


FIGURE 3.3.1-1 PMS VIEWPOINTS.

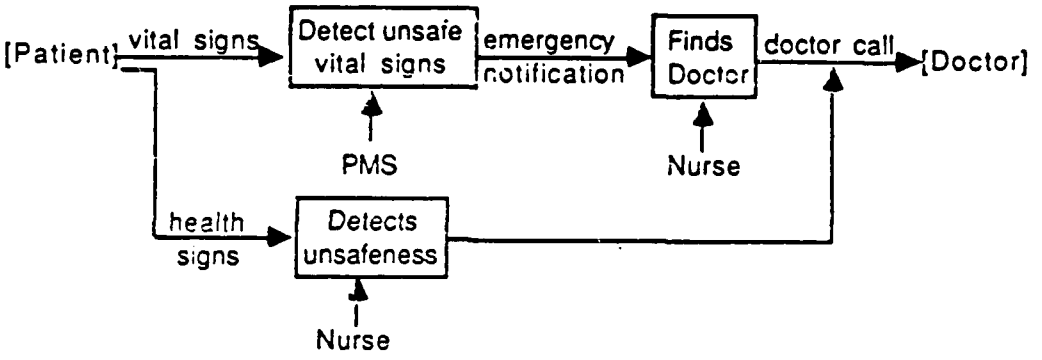


FIGURE 3.3.1-3 DOMAIN MODEL: EXAMPLE TRANSACTION:

"NOTIFY DOCTOR ON UNSAFE CONDITION".

---

1. TRANSACTION "NOTIFY DOCTOR ON UNSAFE CONDITION"

   RELIABILITY = 99% OF NOTIFICATIONS ARE REPORTED
   DURATION = IMMEDIATE
   SOURCE = DOCTOR

2. TRANSACTION "NOTIFY DOCTOR ON UNSAFE CONDITION"

   OPERATIONAL COST < $100/PATIENT/DAY
   SOURCE = HOSPITAL ADMINISTRATOR

3. TRACK PATIENT'S CARE

   SOURCE = NURSE

FIGURE 3.3.1-4 GOALS.

VIEWPOINT PATIENT

ACTIVITY PRODUCES VITAL SIGNS

DERIVES VITAL SIGNS RECEIVED BY PMS.

ACTIVITY GIVES HEALTH SIGNS

USES STATUS CHECK GENERATED BY NURSE.
DERIVES HEALTH SIGNS RECEIVED BY NURSE.

.
.
.

VIEWPOINT PMS

ACTIVITY DETECT UNSAFE VITAL SIGNS

USES VITAL SIGNS GENERATED BY PATIENT.
DERIVES EMERGENCY NOTIFICATION RECEIVED BY NURSE.

.
.
.

VIEWPOINT NURSE

ACTIVITY DETECTS UNSAFENESS

USES HEALTH SIGNS GENERATED BY PATIENT
DERIVES DOCTOR CALL RECEIVED BY DOCTOR.

ACTIVITY FINDS DOCTOR

USES EMERGENCY NOTIFICATION GENERATED BY PMS.
DERIVES DOCTOR CALL RECEIVED BY DOCTOR.

ACTIVITY CHECK PATIENT

DERIVES STATUS CHECK RECEIVED BY PATIENT

.
.
.

VIEWPOINT DOCTOR

ACTIVITY TREAT PATIENT

USES DOCTOR CALL GENERATED BY NURSE.

.
.
.

FIGURE 3.3.1-2 DOMAIN MODEL: ROLE CHARACTERIZATIONS.

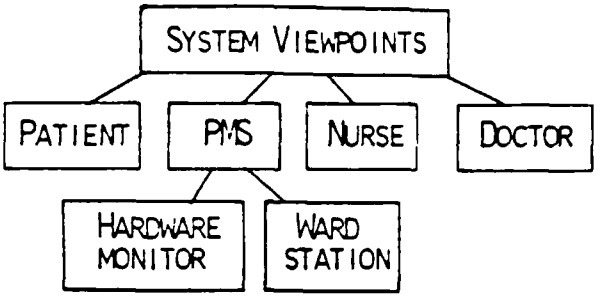


FIGURE 3.3.1-5 FIRST SOLUTION - REFINEMENT OF PMS VIEWPOINT.

VIEWPOINT HARDWARE MONITOR

ACTIVITY DETECT UNSAFE VITAL SIGNS

USES VITAL SIGNS ...
DERIVES UNSAFE CONDITION RECEIVED BY WARD STATION
ATTRIBUTE FREQUENCY = 12 SECONDS
DURATION = 3 SECONDS
OPERATIONAL COST = $500/PATIENT/DAY

VIEWPOINT WARD STATION

ACTIVITY SET ALARM

USES UNSAFE CONDITION GENERATED BY HARDWARE MONITOR
DERIVES EMERGENCY NOTIFICATION RECEIVED BY NURSE.

ACTIVITY PASSES PATIENT REQUEST...

FIGURE 3.3.1-6 FIRST SOLUTION - EXTENDED DOMAIN MODEL

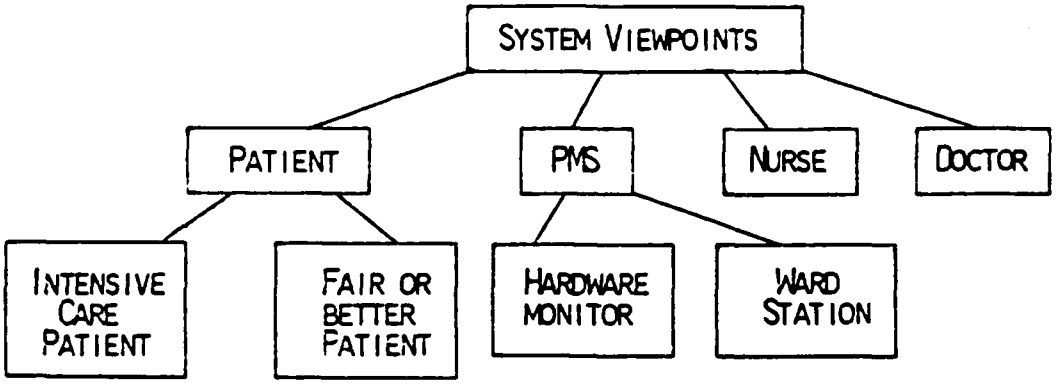


FIGURE 3.3.1-7 REVISED SOLUTION - REFINEMENT OF PMS VIEWPOINTS.

To benefit from consistency and completeness checking, the domain model of figures 3.3.1-2 and 3.3.1-3 (designated here as DM I) can be extended to include figure 3.3.1-6 characterizations. Equivalent PMS activities should then be deleted. We designate the result DM II. In this way, Goals, Requirements, and Solution architectures can be made to share (though different versions of) the same domain model. DM I reflects the domain model referenced by the Goals. DM II has a particular solution "hard-coded" in, but versioning preserves DM I, facilitating tracking across solution revisions and refinements.

Note under activity "detect unsafe vital signs" (figure 3.3.1-6), the inability to easily express which ward station should receive the "unsafe condition" data. The lack of better formalisms leaves the solution's specification ambiguous and appeal must be made to use of free text.

Cost analysis

Analysis of the operational cost must be performed manually. The goal of $100 per patient per day is stated in figure 3.3.1-4. The cost implied by the solution is over $500 per patient per day, because: (1) the solution implies that all patients will be hooked up to a hardware monitor every day of their stay, and (2) this cost alone is $500 per patient per day (figure 3.3.1-6).

Thus another solution is desired, one in which not every patient is connected to a hardware monitor every day of their stay. Perhaps frequent nurse visits could replace use of a hardware monitor most patient days. This suggests the next solution attempt.

Revised Solution architecture and Requirements.

Still attempting to find a feasible solution, the requirements engineer tries this approach:

(1) Patients in intensive care will be hooked to hardware monitors.

(2) Remaining patients, "Fair or better", will be regularly visited by nurses on rounds.

(3) All patients will still have access to a nurse alarm.

(4) PMS, i.e. the Ward station, will inform the nurse of:
A critical situation (1 above),
When it's time to check patients (2 above),
Or of a nurse alarm occurrence (3 above),
providing a form of patient tracking (addressing goal 3, figure 3.3.1-4).

Thus two different kinds of patients must be recognized. This is reflected in figure 3.3.1-7 (PMS functionality will still be allocated between Hardware monitors and Ward stations). Their different needs are characterized in the revised domain model illustrated in figures 3.3.1-8 and 3.3.1-9, in which the solution is again incorporated. Figure 3.3.1-9 replaces the original patient transaction (figure 3.3.1-3) with two transactions, one for each type of patient.

It is assumed that as a result of discussions (with doctor representatives, etc.) on the cost implications of immediate notification and the strategy behind the revised solution, it is determined that the doctor's goal of immediate notification on unsafeness (figure 3.3.1-4) can be relaxed. The result is the two (preliminary) requirements indicated in figure 3.3.1-10, which reference the transactions shown in figure 3.3.1-9.

Analyses on the Revised Solution.

Various analyses can be applied to the solution to gain insight into its appropriateness and feasibility. Some of the analyses will be supported by planned 1990 RET tools, others must be performed manually.

Cost Analysis

Carried out manually. Can goal 2 of figure 3.3.1-4 be met? Let:

I = number of intensive care patients, average per day.
F = number of other patients, average per day.

Assumptions:

(1) Frequency of "Make rounds on patient" = 2 hours (for each fair or better patient).
(2) Time to "Makes rounds on patient" = 2 minutes (per patient).
(3) Time to "Gives health signs" = 1 minute.
(4) Time to "Detects unsafeness" = 2 minutes.
(5) Wage of Nurse = $25 per hour (loaded hourly rate).

These assumptions imply a fair or better patient operational cost of $25 per patient per day (because 12 checks per day times 5 minutes per check (assumptions (2) - (4)) = 60 minutes per day, and 1 hour per day times $25 per hour = $25 per day). Assuming the relative number of patients of each type satisfies:

(6) $F > 5.3 * I$,

the operational cost goal can be met.

VIEWPOINT INTENSIVE CARE PATIENT

ACTIVITY PRODUCES VITAL SIGNS
DERIVES VITAL SIGNS RECEIVED BY HARDWARE MONITOR

... (OTHER ACTIVITIES AS IN PATIENT VIEWPOINT FIG. 3.3.1-2.)

VIEWPOINT FAIR OR BETTER PATIENT

ACTIVITY GIVES HEALTH SIGNS
USES STATUS CHECK GENERATED BY NURSE
DERIVES HEALTH SIGNS RECEIVED BY NURSE

... (OTHER ACTIVITIES (EXCEPT "PRODUCES VITAL SIGNS") AS IN PATIENT VIEWPOINT FIG. 3.3.1-2.)

VIEWPOINT HARDWARE MONITOR

ACTIVITY DETECT UNSAFE VITAL SIGNS
USES VITAL SIGNS GENERATED BY INTENSIVE CARE PATIENT...
...

VIEWPOINT WARD STATION

ACTIVITY PROMPTS FOR ROUNDS
DERIVES ROUNDS NOTIFICATION RECEIVED BY NURSE.

... (OTHER ACTIVITIES AS BEFORE.)

VIEWPOINT NURSE

ACTIVITY MAKES ROUNDS ON PATIENT

USES ROUNDS NOTIFICATION GENERATED BY WARD STATION
DERIVES STATUS CHECK RECEIVED BY FAIR OR BETTER PATIENT.

... (OTHER ACTIVITIES AS BEFORE.)

FIGURE 3.3.1-8 REVISED SOLUTION - EXTENDED DOMAIN MODEL.

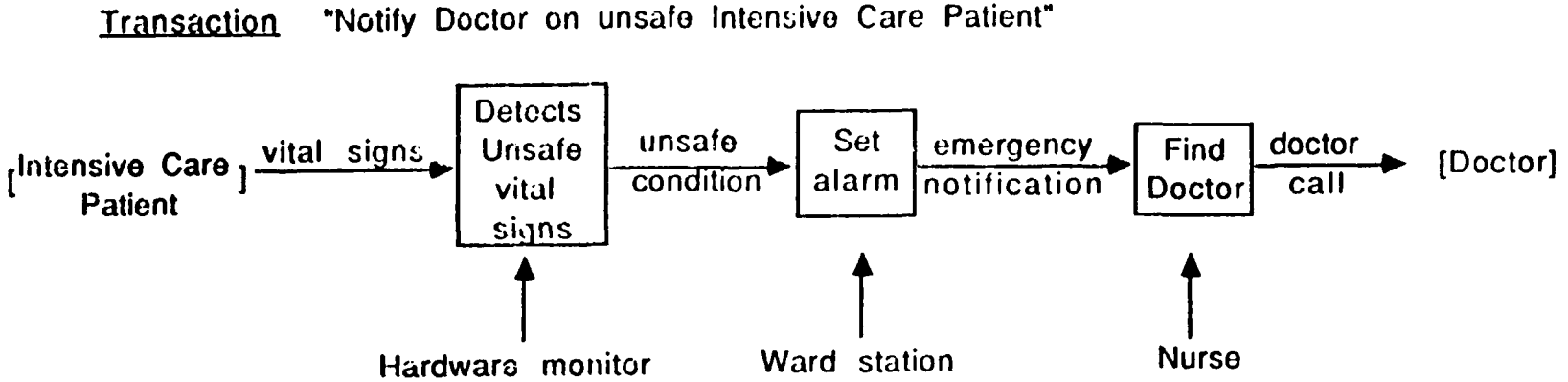
TRANSACTION "NOTIFY DOCTOR ON UNSAFE INTENSIVE CARE PATIENT"

RELIABILITY = 99% OF NOTIFICATIONS ARE REPORTED
DURATION = 30 SECONDS
SOURCE = DOCTOR

TRANSACTION "NOTIFY DOCTOR ON UNSAFE FAIR-OR-BETTER PATIENT"

RELIABILITY = 99% OF NOTIFICATIONS ARE REPORTED
DURATION = 2 HOURS
SOURCE = DOCTOR

FIGURE 3.3.1-10 REQUIREMENTS - EXCERPT SHOWING DOCTOR'S GOAL HAS BEEN REVISED.



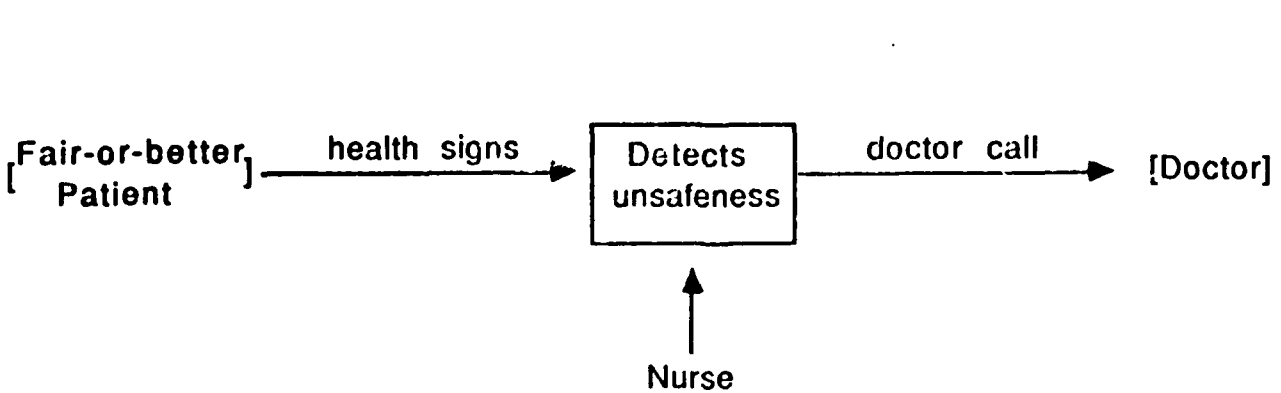


Figure 3.3.1-9 Revised Solution - Extended Domain Model Showing Two Patient Transactions

Performance Analysis

Carried out manually. Can the notification requirements (figure 3.3.1-10) be met? The requirements engineer estimates and allocates time to the activities.
Assumptions:
(1) Time to "Find doctor" (e.g. by beeper) = 10 seconds.
(2) for "Makes rounds on patient": If nurse is given 15 minutes advance warning, the nurse has enough free time to check on a patient within that time. (Note that a check takes 5 minutes according to the cost analysis assumptions.)
Constraints:
(3) Time to "Set alarm" = 5 seconds.
(4) Frequency of "Prompt for rounds" = 2 hours (per patient). Issues "prompt" 15 minutes before patient must next be checked on a round.

Above, (1) and (3) ensure the first notification requirement can be met (sum: maximum time it takes to "Detect unsafe vital signs" (figure 3.3.1-6) + duration of "Set alarm" + duration of "Find doctor" = (12 + 3) + 5 + 10). Above, (2) and (4) ensure the second notification requirement can be met (when function "Prompt for rounds" gives a prompt, (4) says that the nurse has 15 minutes to check the patient on a round, but (2) says that 15 minutes is sufficient to ensure the check takes place).

A Note on Assumptions and Constraints

In the above analyses, "assumptions" are constraints on the PMS environment (e.g. external activities). They could be recorded as attributes or annotations in the domain model. "Constraints" above refers to solution architecture constraints; they thus form part of the solution architecture. It is convenient to have the Solution architecture share the same domain model with Goals and Requirements, and thus such constraints can be recorded as attributes or annotations in the PMS part of the domain model.

Prototyping

Several tools support prototyping. Prototyping addresses the question of whether PMS interface, functionality, and performance requirements are correct (appropriately reflect expected usage and needs). For example, the requirements engineer would probably want Nurse representatives to validate PMS patient-tracking functionality, and perhaps the ward station display as well. Thus he might:

(1) Specify an executable prototype of the ward station to aid validation of the requirements and solution approach to PMS patient-tracking.

(2) Prototype the ward station display to aid validation of the PMS interface display.

(3) Calculate whether the timing constraints on PMS activities can be realized; especially the PMS "constraints" documented during cost and performance analyses.

In the 1990 RET, these activities will be supported by special tools.

For activity (1), figure 3.3.1-11 illustrates a portion of a prototype definition the requirements engineer might construct using the VHLL Prototyping tools. This tool would also assist activity (3).

The definition of the Ward Station in figure 3.3.1-11 is given as a dataflow: arcs represent data, and bubbles represent functions. The dataflow is organized as follows:

* Each Ward Station activity (e.g. "Set Alarm") is represented as an independent horizontal dataflow.

* Each Ward Station state (e.g. "Patient Database") is represented by a vertical grouping of all bubbles that operate on it. The bubbles in such a grouping cannot execute in parallel.

* Labeled boxes represent reference data: boxes labeled "A" represent hardware monitor signals, and boxes labeled "B" represent the patient database. Reference data does not cause activation of bubbles.

* Each bubble has associated code. An annotation beneath the bubble indicates a bound on execution time; for example "< 2" means execution takes less than 2 seconds.

The result is a prototype for activity (1). Ignoring resource contention, we see that the 5-second constraint on "set alarm", obtained by the previous performance analysis, can be met if timer frequency F_1 can be made sufficiently small (e.g. 1 second frequency). This assists activity (3).

For activity (2), figure 3.3.1-12 illustrates the screen portion of an interface the requirements engineer might construct using an RPS tool. Asterisks indicate comments elicited from the nurse when exercising the interface. The RPS tools also assist activity (3).

Stress scenario

Another RPS tool could be used to construct a canned scenario to exercise the proposed solution architecture. Part of a scenario definition is depicted in figure 3.3.1-13.

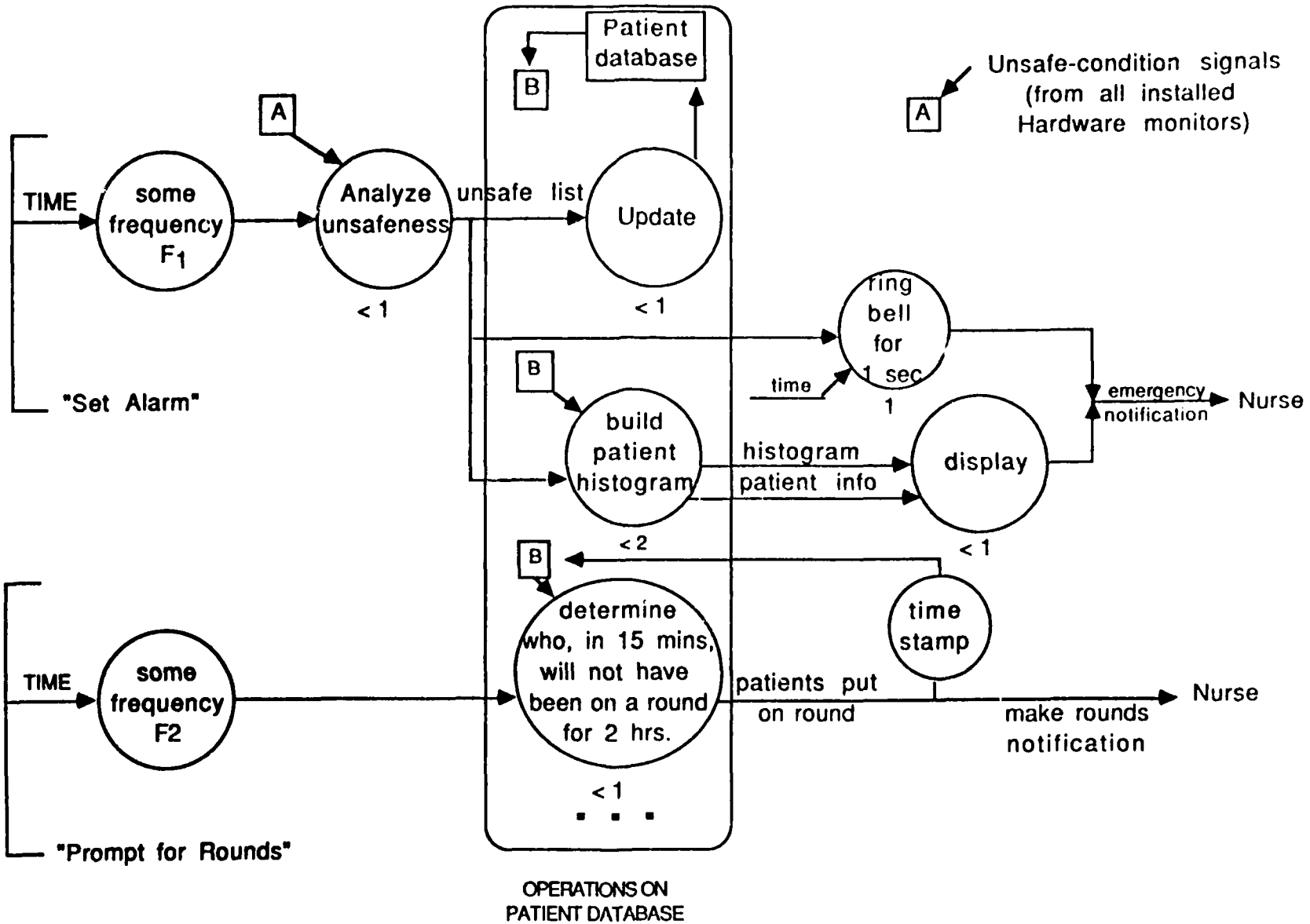
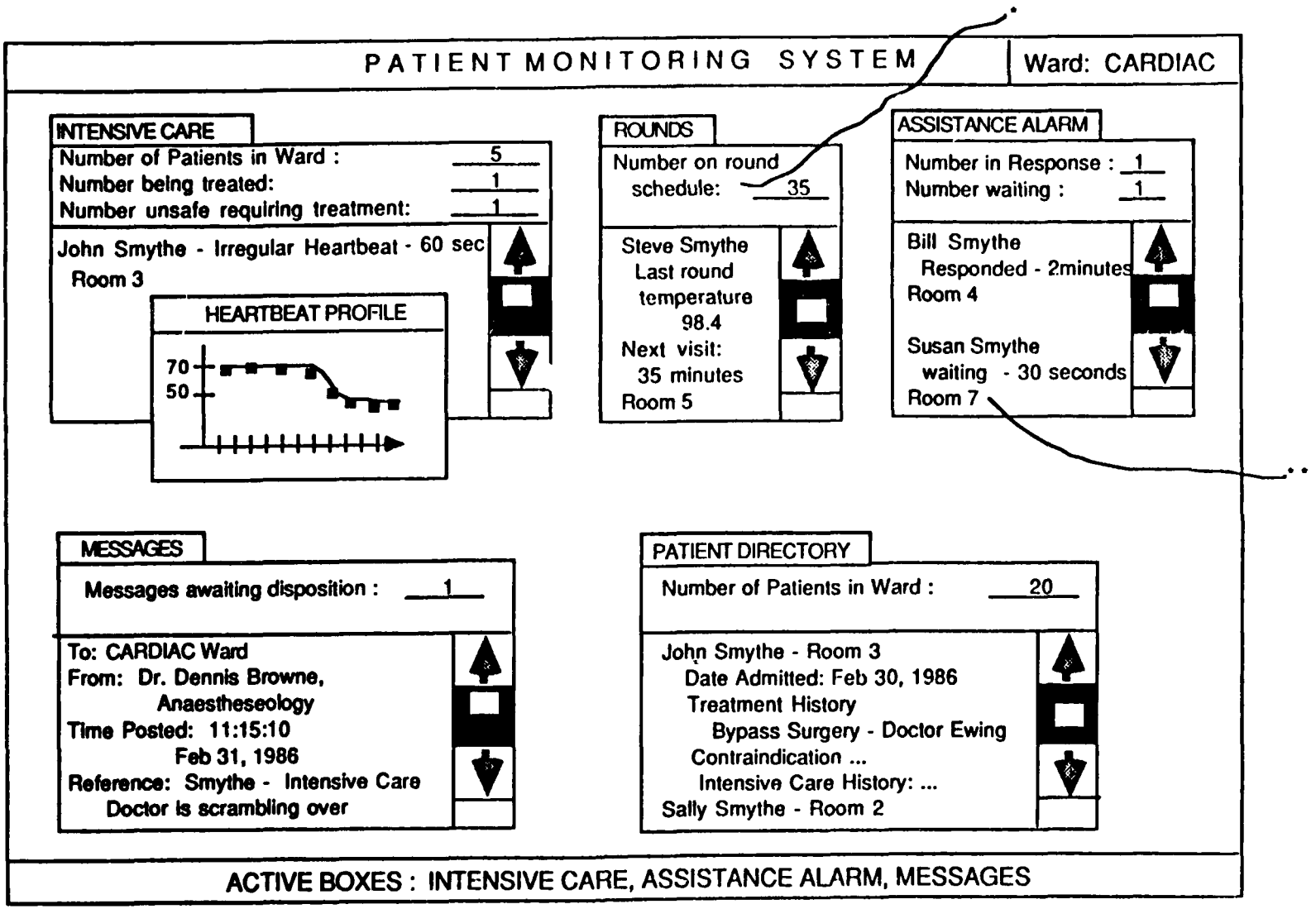


Figure 3.3.1-11 Functional Prototype - Ward Station - an Excerpt.



Comments Elicited from Nurses representative:

***Want a count down bullet: "Number of minutes until next round"**
****Want an indication as to whether patient is in intensive care**

Figure 3.3.1-12 Interface Prototype - Nurse Display Mock-up and Comments Elicited

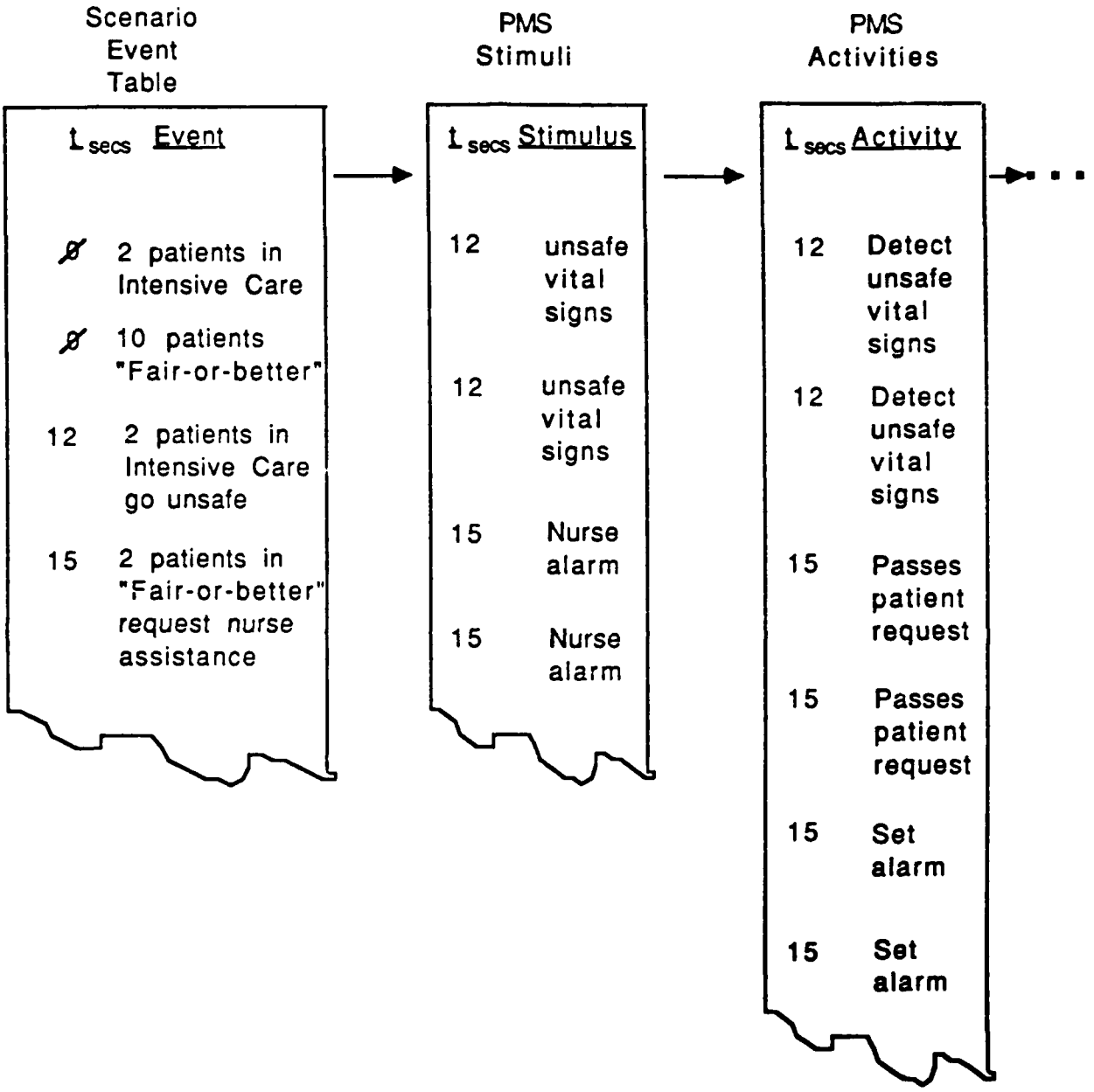


Figure 3.3.1-13 PMS Stress Scenario - Excerpt.

In figure 3.3.1-13, all times are in seconds after the scenario start. In the scenario, two hardware monitors detect unsafe vital signs at 12 seconds. Both take 3 seconds to signal an unsafe condition to the Ward Station. Thus at 15 seconds, the Ward Station must deal simultaneously with two emergency notifications and two Fair-or-better patients' requests for nurse assistance.

Analyses results

The primary result of these analyses is a determination of the soundness of the solution approach.

If fundamentally wrong, a new solution approach must be attempted, or the requirements revised (which might require extensive discussions (both consultation and negotiation) with mission users).

Otherwise, the additional assumptions, constraints, and modifications revealed during the analyses must be added to the Requirements and Solution architecture. Often such additions/modifications will be in the form of added/modified attributes and annotations to their shared domain model (i.e. the version(s) associated with the solution approach).

In our scenario, we assume ideal results of our tool-supported analyses and that no further revisions and analyses are required. However, it is appropriate to include some of the results of these analyses into the Final Requirements and Solution Architecture documentation ("final" as far as the RET exercise is concerned). For example: (1) documentation of all prototyping exercises (objectives, requirements interpretation, design, solution analyses, exercises against scenarios, and results), (2) screen mockups (e.g. figure 3.3.1-12), and (3) stress scenarios for later incorporation into documentation on acceptance and validation testing.

Figure 3.3.1-14 gives an excerpt of the final Requirements and Solution architecture, indicating the changes that need to be made as a result of the new assumptions and constraints revealed during analyses.

The requirements engineer's conclusions on the RET exercise.

What might the requirements engineer conclude from the RET exercise?

* Original goals conflicted, namely low operational cost vs. rapid doctor notification.

* A compromise, achieved thru moderating the rapid doctor notification goal, but not the low cost goal, led to an acceptable set of requirements.

REQUIREMENTS & SOLUTION DOMAIN MODEL

(1) VIEWPOINT NURSE

ACTIVITY FINDS DOCTOR

ATTRIBUTE DURATION = 10 SECONDS

ACTIVITY MAKES ROUNDS ON PATIENT

ATTRIBUTE FREQUENCY = 2 HOURS

DURATION = 2 MINUTES
15 MINUTES ADVANCE WARNING SUFFICIENT TO ENSURE CHECK

ACTIVITY DETECTS UNSAFENESS

ATTRIBUTE DURATION = 2 MINUTES

AVERAGE LOADED WAGE = $25/HOUR.

(2) VIEWPOINT PATIENT

ACTIVITY GIVES HEALTH SIGNS

ATTRIBUTE DURATION = 1 MINUTE

LET I = AVERAGE # OF INTENSIVE CARE PATIENTS PER DAY,

F = AVERAGE # OF FAIR-OR-BETTER PATIENTS PER DAY,

THEN F > 5.3 * I

REQUIREMENTS ON PMS ACTIVITIES

(1) ACTIVITY SET ALARM

ATTRIBUTE DURATION = 5 SECONDS

(2) ACTIVITY PROMPT FOR ROUNDS

ATTRIBUTE FREQUENCY = 2 HOURS

ISSUES THE PROMPT 15 MINUTES BEFORE PATIENT MUST NEXT BE CHECKED ON A ROUND

FIGURE 3.3.1-14 FINAL REQUIREMENTS & SOLUTION ARCHITECTURE - ONLY NEW CHANGES SHOWN.

* PMS functionality and performance requirements were defined and validated.

* A feasible solution approach was defined.

It is important to the RET that these conclusions be carefully documented and linked to the Engineering Context Description. These will aid determination of the effectiveness of RET tools and techniques.

3.3.2 1995 Scenario

The 1995 scenario illustrates these technologies/capabilities:

* A single language for expressing Goals, Requirements and Solution architectures, with these capabilities:

  Shared Domain Model - expressions such as goals, requirements, and solutions can all reference the domain model in order to facilitate their statement.

  Formal Interpretation of Goals and Requirements as predicates against:
  - Solutions - e.g. cost analysis
  - Behavior generated by executing the Solution against scenario(s)).

* Multiple levels of abstraction - there must be a capability of tracking refinements across levels.

* Incremental evolution of requirements.

* Automated generation of scenario - generated from a mission user's outline of the desired behavior.

To summarize, the major theme of the 1995 scenario is that the use of a single formal language permits significant automation of many activities (e.g. cost analysis, comparison of requirements to generated behaviors, scenario generation).

Warning: It is difficult to predict what notations might be employed in 1995. Thus the 1995 scenario freely uses a formal English-like notation to convey to the reader what is being expressed - but not how it might be expressed. It is doubtful that 1995 technology could support such a notation.

Recalling figure 3.3-1 (minus the third wish), we begin the 1995 scenario. As in the 1990 scenario, the hypothetical RET user is called a "requirements engineer".

Deriving Goals

From user interviews and existing documentation, the requirements engineer creates a domain model of the operational environment of the target system (PMS). In 1995, creation of high-fidelity models will be possible. Figure 3.3.2-1 gives only a partial description, illustrating what such a domain model might be like. In the requirements engineering process model, such a domain model is part of (or referenced by) the goals description.

Figure 3.3.2-2 illustrates the goals. The objective of the requirements engineering exercise is to analyze the trade-off between performance and cost wishes. These wishes have now been formally stated as goals. The goals reference information contained in the domain model.

The first goal of figure 3.3.2-2 is really two goals: a logical goal that some doctor be notified and a performance goal that such notification be immediate (the meaning given to "whenever"). To refine the logical goal in order to say which doctor, we need terminology (etc.) that would come from a better domain model of PMS context. Figure 3.3.2-3 illustrates the resulting refined model and refined goal. Figures 3.3.2-1 and 3.3.2-2 represent a high level of abstraction. Figure 3.3.2-3 represents a deeper level of abstraction and better fidelity. By 1995, language mechanisms will exist that permit stating such refinements explicitly (incremental modification) and there will be automatic checks for consistency.

To determine feasibility, the requirements engineer will now treat the goals as preliminary requirements and consider what kinds of solutions are available to him.

Deriving a Candidate Solution Architecture, i.e. Solution

The requirements engineer needs to consider how to build a system that will satisfy the requirements.

Suitable resources (hardware, software, or people) must be identified. For example, consider a "Hardware monitor", figure 3.3.2-4.

The domain model must be extended to include such a resource and constraints on how it might be used so that formal solutions can be expressed. For example, see figure 3.3.2-5, which illustrates how the Hardware monitor is brought into the domain model by extending the latter.

Now the requirements engineer can state the solution. In our example, every patient is hooked to a hardware monitor which is then hooked to his doctor (figure 3.3.2-6).

HARDWARE MONITOR IS A DEVICE;
HOSPITAL TYPE OF INSTITUTION;
PATIENT TYPE OF PERSON,
ADMITTED-TO HOSPITAL;
DOCTOR TYPE OF EMPLOYEE,
WORKS-FOR HOSPITAL;

FIGURE 3.3.2-1 DOMAIN MODEL

---

* WHENEVER UNSAFE (SOME PATIENT),
WARN-DOCTOR (SOME DOCTOR, THAT PATIENT);

* COST <$100/PATIENT/DAY

FIGURE 3.3.2-2 GOALS

HOSPITAL HAS A SET OF WARDS;
PATIENTS ARE ASSIGNED TO A PARTICULAR WARD;
EACH WARD ALWAYS HAS A DOCTOR ON-DUTY;

FOR ANY PATIENT THAT NEEDS A DOCTOR,
USE THE DOCTOR THAT IS ON-DUTY IN THE PATIENT'S WARD.

FIGURE 3.3.2-3 REFINED MODEL AND GOALS

HARDWARE MONITOR IS A DEVICE;
EVERY 15 SECONDS, IT CHECKS THE VITAL SIGNS OF A SINGLE PATIENT;
IT ISSUES AN ALARM IF THE PATIENT IS UNSAFE;
COST IS $500/PATIENT/DAY;

FIGURE 3.3.2-4 A RESOURCE

PATIENT HAS VITAL SIGNS;
HARDWARE-MONITOR SAMPLES VITAL SIGNS;
HARDWARE-MONITOR DETERMINES WHETHER PATIENT IS UNSAFE;
CONNECT HARDWARE-MONITOR ALARM TO DOCTOR, USE TO NOTIFY HIM;

FIGURE 3.3.2-5 EXTENDING THE DOMAIN MODEL TO INCLUDE THE RESOURCE.

FOR EACH PATIENT:

HOOK PATIENT TO HARDWARE MONITOR
HOOK HARDWARE-MONITOR ALARM TO DOCTOR

FIGURE 3.3.2-6 A CANDIDATE SOLUTION

COST = $500/PATIENT/DAY
COST EXCEEDS GOAL

FIGURE 3.3.2-7 AUTOMATIC CALCULATION OF SOLUTION COST AND EVALUATION AGAINST GOALS AND REQUIREMENTS.

Formal analysis of the Candidate Solution Architecture

Extending the domain model to formally include the resource and constraints (as indicated above) facilitates automatic formal analyses of solutions. There are at least two types of analyses that can be done: (1) detection of illegal or incomplete solutions, etc., and (2) determination of operational parameters such as cost, downtime, etc. Formal interpretation of Goals/Requirements as predicates against the results of type 2 analyses lead to an automatic determination of whether the solution is satisfactory in certain ways.

For our PMS example, consider a tool or analysis that automatically calculates operational cost and formally interprets Goals/Requirements as predicates against that cost. Figure 3.3.2-7 illustrates what happens. The resulting operational cost does not satisfy the cost goal (figure 3.3.2-2) and the requirements engineer is so informed.

(Formal analysis continued) Scenario Generation and Analysis

The following will be a 1995 strategy to test a solution for satisfactory performance. It will be largely automated (relative to what can be done even in 1990): (1) create Scenarios, (2) formally interpret Goals/Requirements as predicates, and apply the predicates against the behavior elicited when executing the solution on the scenarios, and (3) note which Goals/Requirements are not satisfied. In the 1995 RET, (2) and (3) will be completely automatic; (1) will be significantly automatic.

Simple scenarios can be automatically generated. In our PMS example, if the requirements engineer requests "Build scenario for a patient", two scenarios would be generated, one for a patient who at some point becomes "unsafe" (see figure 3.3.2-8) and one for an always safe patient. In the general case of scenario generation, the requirements engineer, by indicating key events, will control the combinatorial explosion of scenarios that might be generated from considering all cases. For example, "Build a patient scenario which includes 'patient becomes unsafe'".

In our PMS example, the running of the solution (figure 3.3.2-6) against the scenario (figure 3.3.2-8) produces behavior that can be tested against the predicates resulting from a formal interpretation of the goals (figures 3.3.2-2, 3), yielding the conclusions indicated in figure 3.3.2-9. Recall that the first goal in figure 3.3.2-2 is interpreted as both a logical goal (that notification takes place) and a performance goal (notification must be immediate). Figure 3.3.2-9 says that the notification goal was satisfied, but not the performance goal, because the hardware monitor is a sampling device (figure 3.3.2-4).

```
* PATIENT ADMITTED TO HOSPITAL;
    PATIENT ASSIGNED TO WARD;

* PATIENT HOOKED TO HARDWARE MONITOR;
    DOCTOR HOOKED TO ALARM;

* HARDWARE MONITOR SAMPLES PATIENT VITAL SIGNS;
    PATIENT UNSAFE:
        ALARM ISSUED;
        ON-DUTY DOCTOR WARNED;
```

FIGURE 3.3.2-8 GENERATION OF A SCENARIO DEPICTING AN UNSAFE PATIENT.

```
GOAL OF NOTIFICATION WHEN PATIENT UNSAFE
        ==> SATISFIED

GOAL OF IMMEDIATE NOTIFICATION WHEN PATIENT UNSAFE
        ==> NOT SATISFIED
```

FIGURE 3.3.2-9 AUTOMATIC ANALYSIS OF THE RESULT OF EXECUTING THE SOLUTION AGAINST THE SCENARIO.

The conclusion of this formal analysis is that the solution satisfies the notification goal but not the performance or cost goals. Thus another solution approach must be found that makes these goals realizable, or the goals must be revised.

Deriving Requirements

We assume that further interviews etc. reveal that significant delays in notifying the doctor are acceptable if the patient is noncritical, so the goals can be revised. Such patients' safeness can be monitored by a nurse on rounds. The requirements engineer thus extends the domain model by introducing the concepts of intensive care and nurse, and derives two performance requirements as indicated in figure 3.3.2-10. Note that the performance requirement for intensive care patients has been relaxed to equal the sampling rate of the hardware monitor.

Revising Solution architecture

Next, the requirements engineer attempts to use the nurses to find a solution that makes the requirements feasible, including cost. Parts of such an attempt, solution and extended domain model, are indicated in figure 3.3.2-11. The nurse not only does rounds, but also monitors the hardware alarm.

As with the previous solution (figure 3.3.2-6), the following analysis is done on the solution: cost analysis and a check for satisfiability with the requirements through execution against scenarios. We assume the cost analysis reveals that the current solution satisfies the < $100/patient goal. We also assume that the satisfiability check reveals that introduction of a nurse creates a delay for the intensive care patient case (because there is no longer a direct connection from alarm to doctor). This means we need to find another solution that will make the requirements feasible, or revise the requirements.

Revised Requirements (Demonstration of Incremental Modification)

We assume we revise the requirements. Figure 3.3.2-12 shows the explanation that might be given for the modification to the intensive care 15-second notification requirement.

ALL PATIENTS IN INTENSIVE CARE HAVE HARDWARE MONITORS;
NOTIFICATION OCCURS WITHIN 15 SECONDS;

ALL OTHER PATIENTS ARE MONITORED BY A NURSE ON ROUNDS;
NOTIFICATION OCCURS WITHIN 2 HOURS;

FIGURE 3.3.2-10 REQUIREMENTS

NURSE TYPE OF EMPLOYEE,
WORKS-FOR HOSPITAL;

NURSE MONITORS HARDWARE ALARM;

NURSE PHONES ON-DUTY DOCTOR IF ALARM IS ACTIVATED;

FIGURE 3.3.2-11 CANDIDATE SOLUTION.

TIME OF NOTIFICATION IS RELAXED TO ALLOW TIME FOR NURSE TO PHONE DOCTOR;

FIGURE 3.3.2-12 REVISED REQUIREMENTS - EXPLANATION OF MODIFICATION.

4.0 PANEL RESEARCH AND DEVELOPMENT RECOMMENDATIONS

Objectives of the RET R&D Program

Below, we summarize, by source, the RET R&D program objectives:

* 1990 and 1995 Scenarios - the R&D program should provide tools and methods that work together in the ways illustrated by the scenarios (section 3.3). The tools and methods should have the capabilities illustrated by the scenarios.

* Requirements Engineering Process Model - The R&D program should provide tools and methods to support process model activities (sections 3.1 and 3.2; detail is found in appendix C).

* RADC's plans and goals - The R&D program should help fulfill these goals for the RET (section 2): (1) The RET should support evaluation of the effectiveness of tools and methods. (2) The RET should make a full range of requirements engineering capabilities accessible to Air Force mission users and acquisition engineers. (3) The RET should host the currently-contracted tools, and by 1990, they should be integrated.

* Long-range architecture for the RET - The R&D program should realize the panel's vision of an RET architecture featuring (appendix D): (1) a direct manipulation-style user interface to all objects, (2) a database serving as the common repository for all requirements-related information, and (3) a formal language for expression of goals, requirements, and solution architectures.

  In the long term, all RET tools and methods should be structured to fit this architecture.

References are made to these objectives in the sections that follow. Section 4.1 summarizes the panel's strategy for obtaining an integrated RET. Section 4.2 discusses an R&D program consisting of two tracks: (1) an Evolutionary Track for developing tools and methods that in the near term provide the best payoff in better requirements; and (2) a Formal Language Track for exploring the higher risk/payoff implications of a formal requirements language. Section 4.3 discusses the panel's recommendation on the relative allocation of R&D resources between these two R&D tracks, and the relative prioritization of issues within each track.

4.1 Near-term Integration of RET

To address RADC's objective of integrating the currently-contracted tools, the panel recommends that integration be

achieved by having the tools work off a common database and be accessed through a common user interface. This level of integration means that: (1) tools can share data, and (2) the RET user is given uniform access to tools and their data and is free to invoke tool functions in an order natural to his/her application.

This approach will produce an early version of the long-range RET architecture.

An integrated RET will also help in the evaluation of tools; for example, by providing the basis for a broader range of control experiments.

To significantly reduce the amount of effort required to achieve integration, the panel recommends a near-term strategy of standards and cooperation between the RADC tool contractors. The integration strategy is further discussed in appendix D.

To provide RET users some of the benefits of integration in the very near term, the panel recommends a loose coupling of the currently-contracted tools. The loose coupling plan is also discussed in appendix D.

4.2 Requirements Engineering Testbed (RET) Research and Development Program

4.2.1 Two-Track Program

To meet the objectives stated at the beginning of section 4, the panel identified two themes on which RET R&D program efforts should focus: (1) providing near-term support for these activities: prototyping, requirements analysis, and evaluation of tools; and (2) a formal treatment of requirements. The R&D program consists of two tracks to deal with these two themes. Figure 4.2.1-1 depicts the two themes and their relationships. Below, we expand on these themes and their associated activities and then discuss the figure.

Theme of the Evolutionary Track

Both the 1990 scenario and the process model characterization demonstrate the importance of prototyping and the role of scenarios in driving prototypes. (For examples, refer to figures 3.3.1-11, 3.3.1-12, and 3.3.1-13.) Prototyping gives a mission user "visibility" into specifications of system and software requirements by helping the mission user determine whether his/her needs are being addressed. Thus the panel recommends that the "creation of prototypes and scenarios, and analysis of results" activity be an early focus of the RET R&D program.

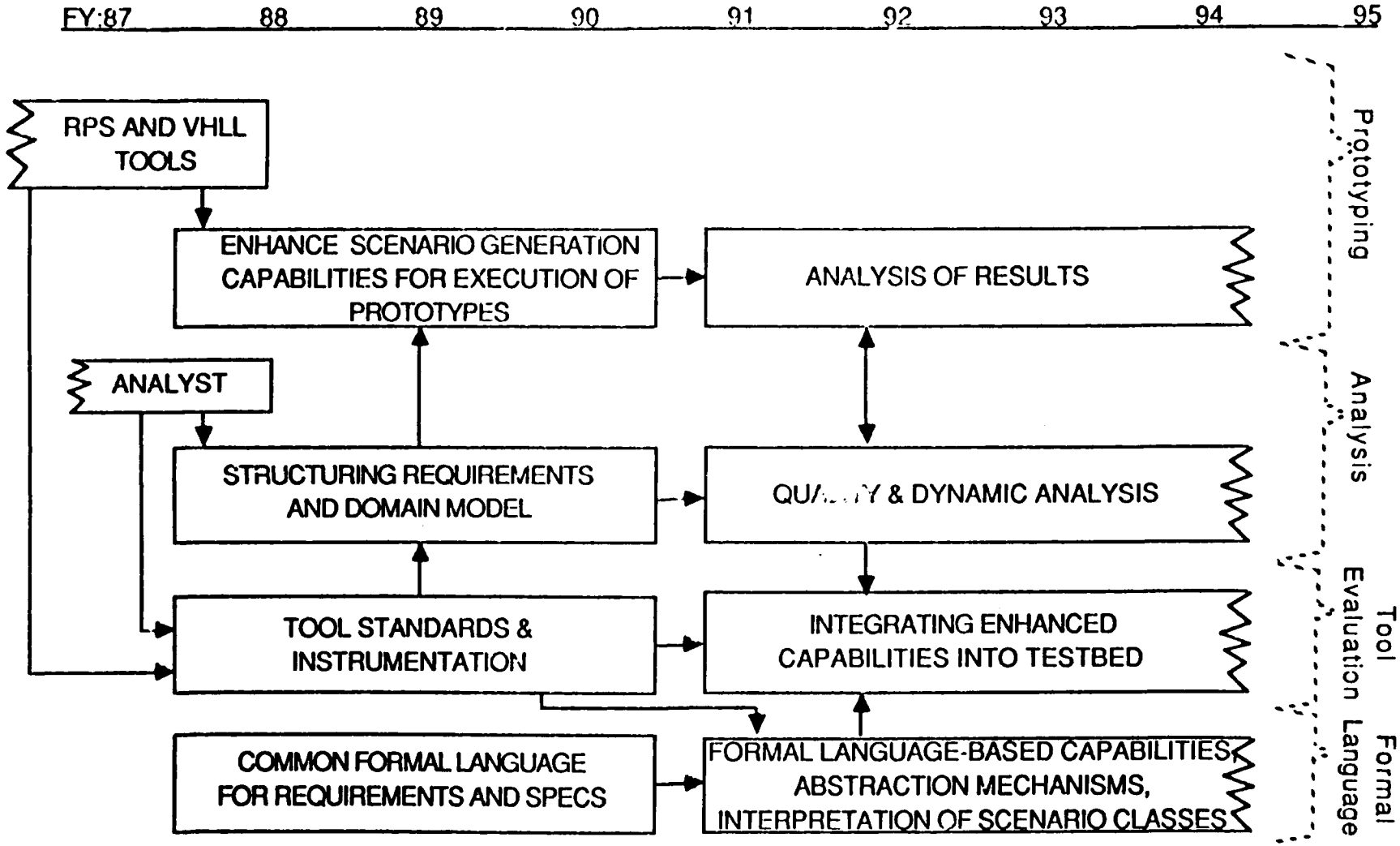


Figure 4.2.1-1 RET R&D ROADMAP

Some of the objectives stated in section 4 imply the need for tools and methods that address the non-solution-architecture phases of requirements engineering; specifically, goals and requirements synthesis and analysis. Such tools and methods would help mission users state their needs and decisions at the mission level. Both scenarios demonstrate the need for such capabilities. Such a need must be satisfied in the near term. Thus, the panel recommends that the "analysis on requirements" activity be another early focus of the RET R&D program.

Evaluating the effectiveness of RET tools and methods requires the capability to track their use and collect results. An integrated RET would help control independent parameters (e.g. style of presentation, format of input data), a prerequisite for parallel experiments. Thus the panel recommends that the "measurement of tools in an integrated RET" activity also be an early focus of the RET R&D program.

To successfully provide near-term support for the three activities above requires a low risk and early payoff strategy. In the long term, the resulting RET capabilities would be enhanced/refined. The panel organized an "Evolutionary Track" of R&D efforts to do this. The Evolutionary track would provide the capabilities illustrated in the 1990 scenario and support most process model activities. The Evolutionary Track is described in section 4.2.2.

The Theme of the Formal Language Track

The panel recognized early on that representing requirements in a formal language was a high-payoff approach, but such an approach would fail to provide near-term solutions to the R&D program objectives. Nevertheless, such an approach would address these objectives not being addressed in the other track: (1) help automate requirements traceability and assessment of requirements coverage, and (2) provide a language for representing requirements, solution architectures, and goals; a major element of the long-range RET architecture. The panel thus defined a "Formal Language Track" whose focus would be to provide such a formal language.

The Formal Language Track would also provide the capabilities illustrated in the 1995 scenario and support most process model activities.

The Figure

Figure 4.2.1-1 depicts 1990 and 1995 goals of the two themes: creation of prototypes and scenarios and analysis of results ("Prototyping"), analysis on requirements ("Analysis"), measurement of tools in an integrated RET ("Tool Evaluation"), and formal language. Their dependencies with each other and with the currently-contracted tools are indicated by the arcs.

Analyst tool capabilities are the basis for the 1990 Analysis capabilities for structuring requirements and the domain model. Domain models provide necessary information for building scenarios and simulations, thus the vertical dependency with 1990 Prototyping goals. 1990 Prototyping goals are also dependent on the prototyping and scenario generation capabilities provided respectively by the VHLL and RPS tools.

1995 Prototyping goals extend 1990 Prototyping capabilities by providing capabilities for sensitivity analysis on requirements. 1995 Analysis goals extend 1990 Analysis capabilities by providing capabilities for dynamic analysis and quality critiquing. Various interactions are possible between 1995 analysis tools for these two activities, hence the double-headed vertical arrow.

The 1990 Tool Evaluation standards will be influenced by the standards adopted by developers of the currently-contracted tools. These 1990 standards will in turn guide all subsequent RET tool development. The 1995 Tool Evaluation goal is to integrate these new Analysis, Prototyping, and Formal Language capabilities into the instrumented RET to facilitate their evaluation.

The 1990 Formal Language goals are independent of 1990 Analysis and Prototyping efforts and the currently-contracted tools. The 1995 Formal Language goals include incorporating abstraction mechanisms into the language and interpreting classes of scenarios. These capabilities must also be integrated into the RET.

From the perspective of milestones, the 1995 RET will be a mature experimental facility, hosting matured analysis, prototyping, formal language, and evaluation capabilities. The 1990 RET will be a prototype of the 1995 RET, still integrated and instrumented for evaluation, but featuring only a few mature tools, in particular, the currently-contracted tools.

4.2.2 Evolutionary Track

As explained above, the theme of the Evolutionary Track focuses on three activities. Below, in section 4.2.2.1, we consider these activities in turn, indicating, by R&D roadmap, the panel's strategy of achieving the theme's 1990 and 1995 goals. In section 4.2.2.2, we consider the R&D issues that constitute the Evolutionary Track.

The linkage between section 4.2.2.2 R&D issues and the R&D effort boxes of the roadmaps in section 4.2.2.1 is indicated by a code(s) in the lower right-hand corner of most boxes in the roadmaps. This code(s) indicates the R&D issues of which the R&D effort is a part. Below, we indicate the correspondence between R&D issue names and codes:

DM: Domain Models and Information
RSA: Requirements - Static Analysis
RA: Requirements Analysis Methodology
DA: Dynamic Analysis
SG: Scenario Generation Support & Scenario Coverage Analysis
AR: Scenario Execution and Analysis of Results
EC: Estimation of Cost, Risk, Time in System Development; Performance & Execution Costs Analysis
RE: Requirements Evaluation
TE: Testbed Effectiveness
UI: User Interface
DB: Database
ETI: Evolutionary Testbed Integration

4.2.2.1 Objectives and Roadmaps

Requirements Analysis

The goal of Requirements Analysis is to extend and automate capabilities in the analysis of requirements. Analyses include analysis for consistency and completeness, analysis of interrelationships among the requirements and with domain models and scenarios, and quality assessment: detection of redundancy, determination of understandability and modifiability.

The parts of the 1990 scenario addressed include:

Capabilities for partially formal characterizations of: the problem domain (including resource models), functional requirements, scenarios, and some nonfunctional requirements. Also, static analysis capabilities on these.

Capabilities for expressing goals.

Capabilities logically equivalent to those indicated in figures 3.3.1-1 through 3.3.1-10, the Scenario Event Table in figure 3.3.1-13, and figure 3.3.1-14.

Figure 4.2.2.1-1 indicates the panel's strategy of meeting the goal of this activity. The roadmap indicates: (1) the formalisms, tools, methods, etc. that support the above-mentioned (and longer range) capabilities, (2) the corresponding R&D efforts that will produce these formalisms, tools, methods, etc., and (3) the dependencies between these.

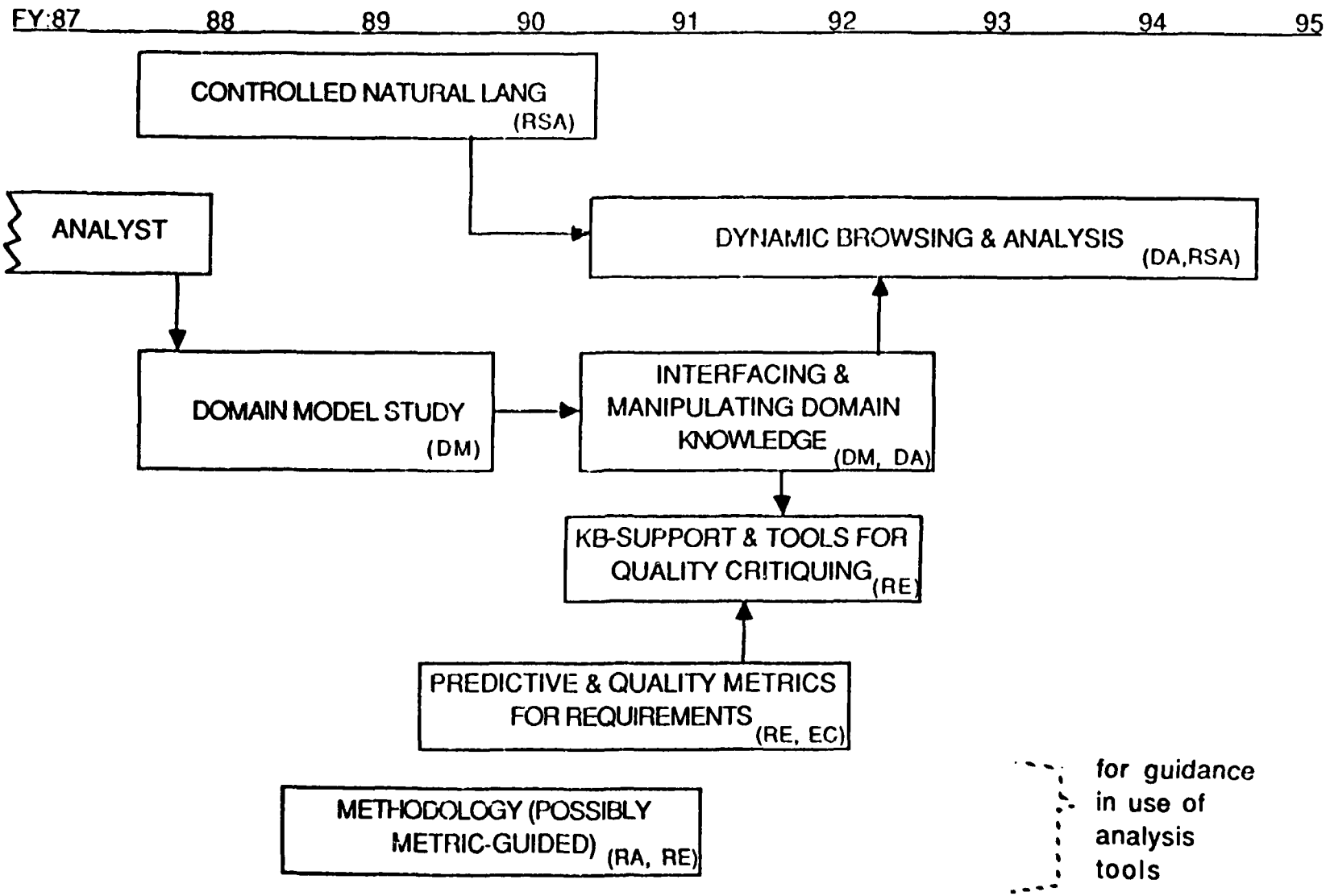


Figure 4.2.2.1-1 EVOLUTIONARY TRACK ROADMAP: R&D EFFORTS TO AID MAKING BETTER ANALYSIS ON REQUIREMENTS

Prototyping

The goal of Prototyping is to extend and automate capabilities in the creation of prototypes and scenarios for experiments, and analysis of experimental results. Related issues include: creation of support simulations, coverage analysis, and the presentation of experiment-generated data.

The parts of the 1990 scenario addressed include:

> Capabilities for partially formal characterizations of: scenarios and their interfaces (e.g. inputs) to prototypes.
>
> Capabilities for building executable models, performance models, and scenarios. Capabilities for exercising prototypes against scenarios. These capabilities provide an essential basis for evaluating feasibility and trade-offs.
>
> Capabilities logically equivalent to those indicated in figures 3.3.1-3, 3.3.1-9, and 3.3.1-11 through 3.3.1-13.

Figure 4.2.2.1-2 indicates the panel's strategy of meeting this activity's goal. (In the figure, two boxes have wavy left edges. Such a left edge is used to indicate that the true position of the edge lies further to the left but can't be shown without creating an overlap or impacting the figure's compactness. The true position can be found by referencing the corresponding R&D issues in Appendix E.)

Tool Evaluation

The goal of Tool Evaluation is to provide capabilities for measuring the effectiveness of RET tools in terms of improvements to processes (e.g. productivity) and products (e.g. quality). A related goal is to provide an integrated RET in which the best features of each tool can be applied to the same set of requirements.

The parts of the 1990 scenario relevant to tool evaluation include:

> Establishing RET exercise objectives, documenting results of the RET exercise.
>
> The tool evaluation context might appear in the Engineering Context Description, establishing objectives for and constraining the RET exercise for the RET purpose of tool evaluation.

Figure 4.2.2.1-3 indicates the panel's strategy of meeting this activity's goal.

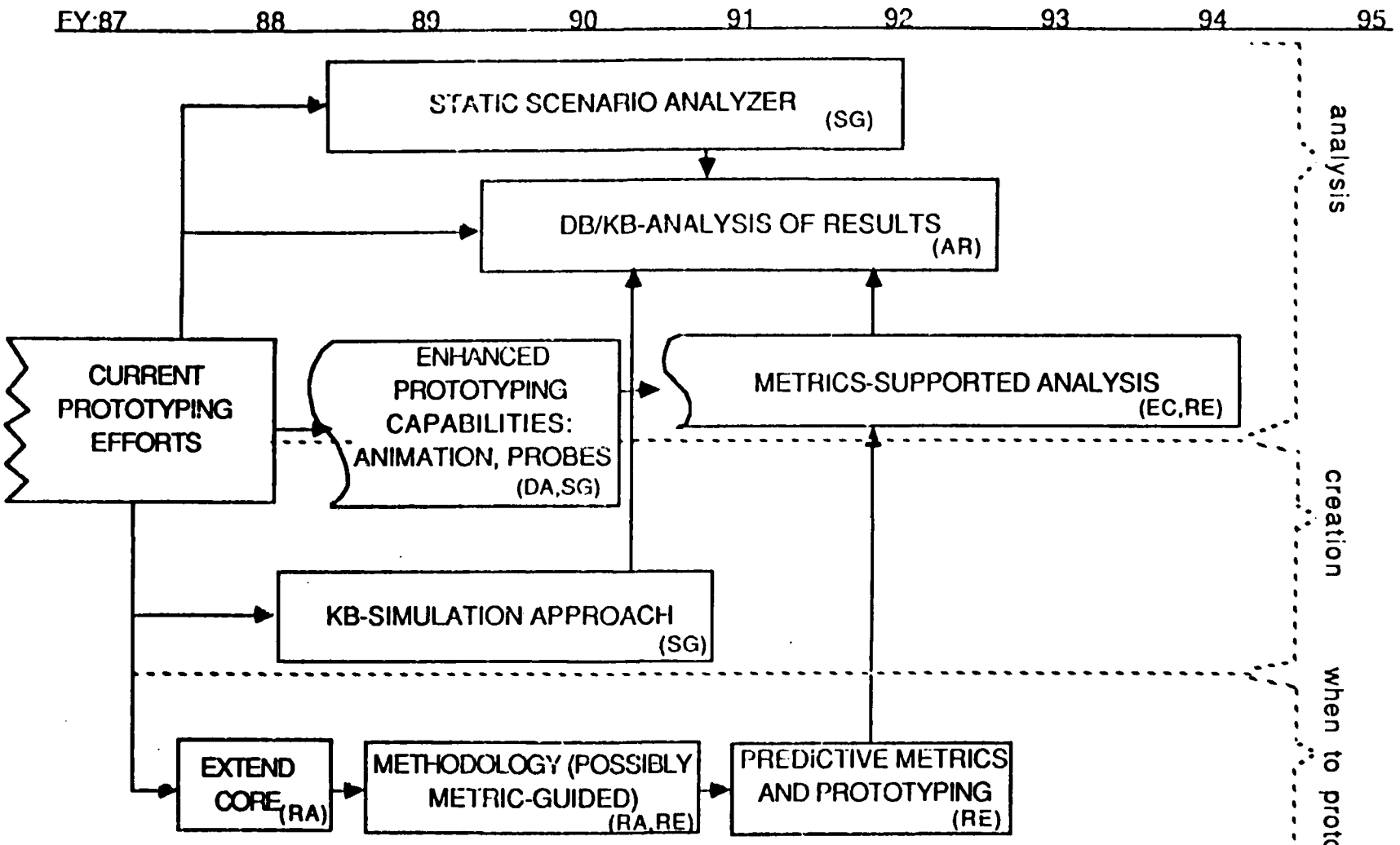


Figure 4.2.2.1-2 EVOLUTIONARY TRACK ROADMAP:
R&D EFFORTS FACILITATING CREATION OF PROTOTYPES & SCENARIOS
AND ANALYSIS OF RESULTS

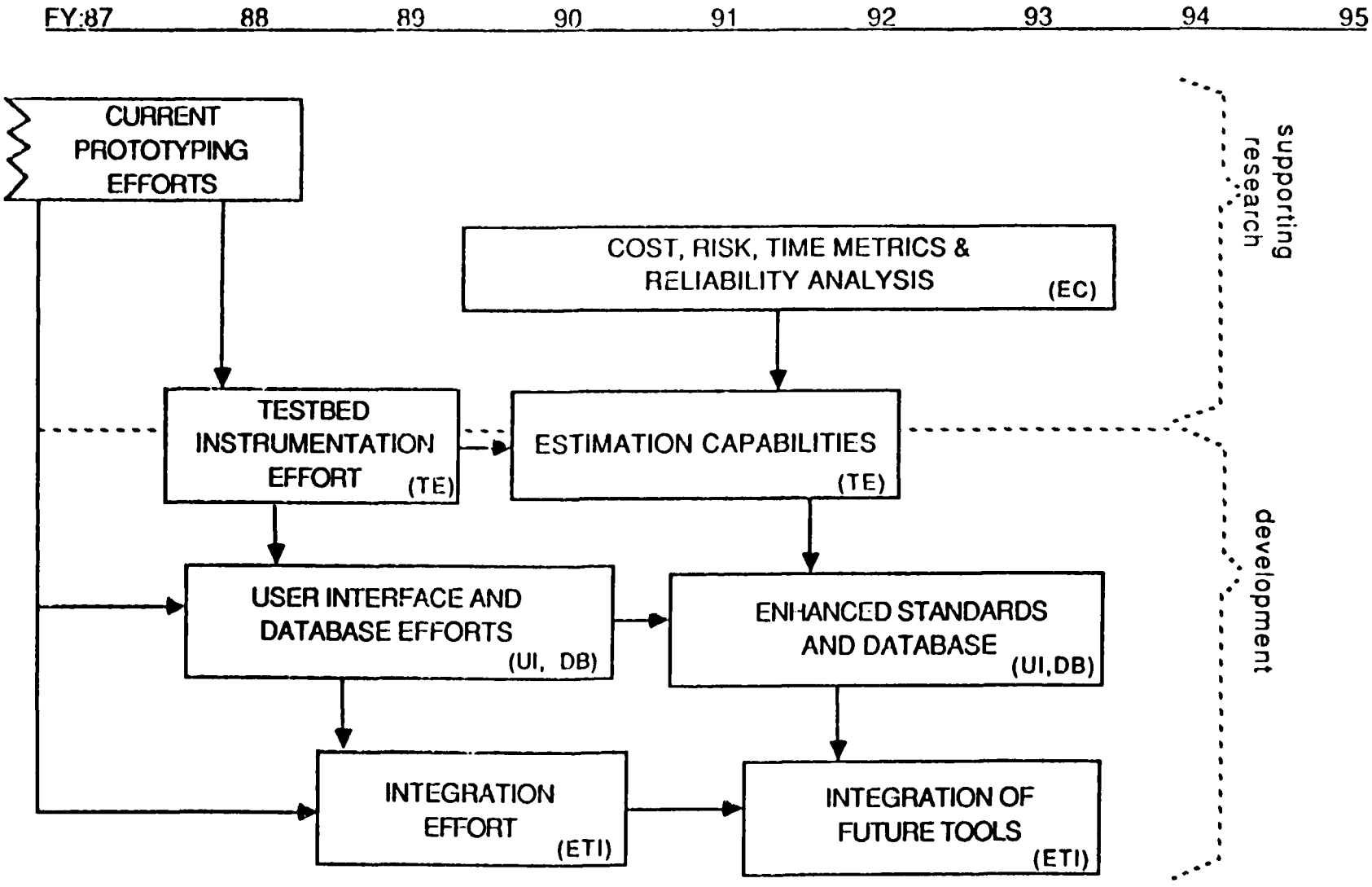


Figure 4.2.2.1-3 EVOLUTIONARY TRACK ROADMAP: R&D EFFORTS FACILITATING MEASUREMENT OF TOOLS IN AN INTEGRATED TESTBED

4.2.2.2 Research and Development Issues

The Evolutionary Track consists of both research and development issues. Appendix E gives detailed characterizations of all Evolutionary Track issues. Below, we provide a summary of the research issues followed by a summary of the development issues.

Characterization of Evolutionary Track Research Issues

The research issues generally correspond to activities in the requirements engineering process model. This is because the panel used the process model to help identify the R&D issues that would help realize desired RET capabilities.

The Evolutionary Track R&D issues are considered here in an order roughly corresponding to a top-down walk-through of the process model.

Each research issue is characterized in terms of: where it fits in the process model, what are the research objectives, what is the recommended solution approach, and recommendations regarding funding priority based on assessed risk and payoff. The funding priority scheme assigns each issue a priority of High, Medium-high, Medium, Medium-low, or Low. A summary of the prioritization is given in section 4.2.4.1.

GOALS AND REQUIREMENTS SYNTHESIS

Where issue fits in process model: In transforming wish lists into goals and transforming goals into requirements.

Research Objectives: Provide capabilities for: (1) expressing and viewing goals and requirements and (2) expressing goals and requirements in one or more domain-specific languages and integrating the different expressions.

Solution Approach: Develop mechanisms for syntax-directed editing, view management, and for supporting reuse. Enhance the Analyst to interface and utilize these mechanisms.

Recommendation: Funding priority: Medium-low. To some extent solution mechanisms can be provided by database technology. These should be brought into the RET; in which case the probability of success is high, and no specific research funding is required.

DOMAIN MODELS AND INFORMATION

Where it fits in process model: Domain information is essential to development of goals, requirements, and solution architectures.

Research Objectives: Provide capabilities to collect and organize domain information and make it accessible to tools and requirements engineers.

Solution Approach: Select and develop mechanisms for capturing and structuring domain information, e.g. knowledge acquisition and domain-specific languages/interfaces. Build interfaces for tools to exploit domain information. Investigate role for expert systems that use domain information to help elicit and validate requirements.

Recommendation: Funding priority: Medium-high. Risk: low to moderate. This research offers much promise, but many of the capabilities it attempts to provide will remain manual.

REQUIREMENTS - STATIC ANALYSIS

Where it fits in process model: Static analysis on requirements.

Research Objectives: Provide consistency and completeness checking capabilities.

Solution Approach: Define a controlled (restricted vocabulary) natural language for the expression of requirements. Develop checkers that take such requirement expressions as input and determine their consistency, completeness, etc. with respect to meta-knowledge bases.

Recommendation: Funding priority: Medium. Risk: low. This research has some promise, but the approach is fairly independent of other R&D efforts.

REQUIREMENTS ANALYSIS METHODOLOGY

Where it fits in process model: Guides the order in which process model activities are carried out.

Research Objective: Develop a methodology spanning all requirements engineering activities that will provide guidance in the use of new requirements analysis capabilities.

Solution approach: Near term: Extend the CORE method into solution architecture synthesis and analysis and provide guidance in use of RPS and VHLL Prototyping tools. Long term: Select a then-existing methodology that better meets the research objective based on effectiveness evaluation of existing tools.

Recommendation: Funding priority: Medium-high. Risk: Moderate. Tools must be evaluated to determine their range of effectiveness. This research has much promise.

DYNAMIC ANALYSIS

Where it fits in process model: In dynamic analysis and rapid prototyping.

Research Objectives: Provide the user with an animation-based capability (i.e. graphically depicting flow of information and control) to investigate the consistency, completeness, and validity of a set of requirements.

Solution Approach: Enhance prototyping capabilities with: (1) the ability to run animated exercises of the proposed system, (2) the ability to graphically browse through requirements interrelationships, and (3) a knowledge-based enhancement: ability to represent and analyze requirements against a known domain model.

Recommendation: Funding priority: Medium-high. This research offers much promise at generally low risk. Payoff and risk of knowledge-based enhancements is not yet clear.

SOLUTION ARCHITECTURE SYNTHESIS

Where it fits in process model: Design candidate architecture.

Research objectives: Develop a design assistant that will help construct and/or critique a solution architecture.

Solution approach: Develop meta-models of design goodness and enhance prototyping tools with knowledge base support for creation of good designs.

Recommendation: Funding priority: Low. Risk: high. Wait for basic results from other investigations into the "design problem".

SCENARIO GENERATION SUPPORT & SCENARIO COVERAGE ANALYSIS

Where it fits in process model: Rapid Prototyping (building scenarios that drive prototypes and determine coverage).

Research objectives: Provide the capability to build adaptive scenarios and support-simulations that drive a prototype. Provide the capability to determine which parts of the prototype (and associated requirements) were brought into play during execution.

Solution Approach: Provide a knowledge-based simulation system that can interface and run executable and performance models of the target system, simulating the interactions between the scenario and the models. Develop dynamic probes into a prototype. Investigate coverage by proving the scenario from the prototype description.

Recommendation: Funding priority: High. Risk: low for scenario generation, higher for coverage. Resulting scenarios should be able to work with all prototyping tools.

VALIDATION OF PROTOTYPE AND SCENARIOS

Where it fits in process model: Rapid prototyping.

Research Objectives: Provide the capability to validate the prototype and scenario for consistency, completeness, and logical correctness. Provide the capability to validate the results of executing the prototype against the scenario.

Solution approach: Build knowledge-based syntax and semantics checker for prototypes and scenarios. Create a library of metered and validated prototypes that would be used to gauge the accuracy of the results of a particular prototyping exercise.

Recommendation: Funding priority: Low. Much of the necessary groundwork (i.e. reusability, formal expression of prototype and scenarios) for validation is being addressed by other research components of the track. Manual validation methods will continue to be necessary.

SCENARIO EXECUTION AND ANALYSIS OF RESULTS

Where it fits in process model: Rapid Prototyping.

Research objectives: Provide tools to collect, analyze, and present the data generated during prototype experiments.

Solution Approach: In the near-term: a database to hold results and data management facilities to aid human analysis; each prototyping tool will provide a specialized analysis capability. In the long-term: knowledge-based aids for evaluation of prototype sensitivity.

Recommendation: Funding priority: High. Risk: near-term: low; long-term: moderate to high. Long-term effort assumes an integrated RET in which one scenario can drive multiple models and results can be correlated.

ESTIMATION OF COST, RISK, TIME IN SYSTEM DEVELOPMENT; PERFORMANCE & EXECUTION COSTS ANALYSIS

Where it fits in process model: Analysis of performance and reliability, and development cost, risk.

Research objectives: Provide metrics-based capabilities for the estimation of cost, time and performance as a basis for making trade-offs and doing impact analysis.

Solution approach: Provide metrics for cost, time, risk, and project size, and provide related tools that do the measurement and analysis. Add metrics for distributed/knowledge-based systems. Develop tools that do fault-handling analysis and reliability estimation.

Recommendation: Funding priority: High. Risk: Moderate. Metrics are often subjective, thus special attention is needed. Research has great promise supporting critical trade-off and impact analysis activities.

REQUIREMENTS EVALUATION

Where it fits in process models: Primarily requirements evaluation and reformulation, but also static analysis, solution architecture synthesis and prototyping.

Research objectives: Provide capabilities to determine requirements quality and compare alternative sets of requirements and solution architectures. Provide an associated methodology that helps focus requirements engineering activities to produce better estimates of system time, cost, and performance.

Solution approach: Develop a Metric Guided Methodology with: (1) tools for quality assessment, (2) knowledge-based tools for acquiring and manipulating knowledge of the system being developed, (3) support and correlate different design representation schemes, and (4) combine predictive metrics with prototyping.

Recommendation: Funding priority: Medium. Risk: Moderate. There will be difficulties in validating metrics and in the knowledge-based aspects of the work, but if these difficulties can be overcome, the payoff will be high.

TESTBED EFFECTIVENESS

Where it fits in process model: It relates to the engineering context description.

Research objectives: Provide the capability to determine the effectiveness of new Requirements Engineering Testbed tools and techniques.

Solution Approach: Instrument the RET for time/effort and resource utilization measurements. Provide basis (metrics, prototyping) for development cost/risk/schedule estimation. Compare requirements quality before/after application of a tool or technique.

Recommendation: Funding priority: Medium-low. Mostly a development concern. This effort might be incorporated in the "Evolutionary Testbed Integration" development issue.

Characterization of Evolutionary Track Development Issues

There are three development issues: User Interface, Database, and Evolutionary Testbed Integration. All three issues form part of the panel's strategy for: (1) integrating the RET, and (2) instrumenting the RET as a basis for tool evaluation. The "User Interface" and "Database" issues also provide initial versions of corresponding elements in the long-range RET architecture.

The development issues are generally given the same kind of characterizations as were the research issues, except that they are given a characterization of how they fit in the long-range RET architecture rather than of how they fit in the process model.

USER INTERFACE

How it fits in RET architecture: The use of consistent interfaces by all tools eases user access to tool and data. Supports tight integration of tools.

Development objectives: All tool developers should be required to use a consistent approach to end-user communication.

Solution Approach: (1) Establish user interface models and standards to be observed in the development of all tool interfaces and in the use of run time support packages. (2) Check compliance with standards by all tool contractors. (3) Evolve standards.

Recommendation: This effort should be initiated very early in the R&D program so as to affect all development activities.

DATABASE

How it fits in RET architecture: Provide data storage and data management capabilities to support tight integration of RET tools via shared data.

Development objectives: Provide viewing/reporting, editing, classification, and export/import facilities to aid RET users in accessing and managing large volumes of complex data.

Solution Approach: Select a general-purpose DBMS which is efficient in the storage of design objects and which provides the proper facilities. Develop common data object descriptions and conversion routines so existing tools can share data. Customize data management facilities for RET needs.

Recommendation: This is a high-payoff low-risk component of the RET, and critical to the evolutionary track.

EVOLUTIONARY TESTBED INTEGRATION

How it fits in RET architecture: This effort would support tight integration of existing tools, allowing the use of key capabilities of each tool on the same set of requirements.

Development objectives: Develop an integrated RET featuring: an RET experimentation methodology, a common database and user interface for tools, and a way of bringing new tools in.

Solution Approach: Identify the degree of tool interaction and tracking desired. Assuming the "database" and "user interface" efforts have progressed, begin modifying the tools. For tightening the integration, identify which tools should be invoked by data changes and which should be explicitly invoked by the user. Develop an RET experimentation methodology.

Recommendation: This effort is critical to achieving tool interaction and also for RET effectiveness assessment. Possibly incorporate instrumentation capabilities from "Testbed effectiveness" into this effort.

4.2.3 Formal Language Track

As explained in section 4.2.1, the theme of the Formal Language Track is to provide a formal treatment of requirements. In section 4.2.3.1, we indicate, by an R&D roadmap, the panel's strategy of realizing this theme. This roadmap also shows dependencies between all R&D issues of the track.

In section 4.2.3.2, the research issues which constitute the Formal Language Track are characterized.

4.2.3.1 Objectives and Roadmap

The goal of the formal language track is to provide a formal treatment for requirements in which automated support can be given for tracking requirements into specifications and checking for requirements satisfaction when a specification is run against scenarios.

The addressed parts of the 1995 scenario include:

> All parts are addressed: formal support will be given to all illustrated synthesis activities and analysis activities except for synthesis of goals, for which a methodology will be provided.

Figure 4.2.3.1-1 indicates the panel's strategy for providing a formal treatment for requirements. Each box represents a research issue of the Formal Language Track. (Because of the close correspondence between Formal Language Track R&D efforts and R&D issues, there is no need in the roadmap for a special linkage such as that employed in the roadmaps of section 4.2.2.1.)

4.2.3.2 Research Issues

The Formal Language Track consists only of the research issues indicated in figure 4.2.3.1-1. Appendix E gives detailed characterizations of these issues. These issues can be considered as belonging to one of two types: (1) issues that directly contribute to and/or are strongly dependent on a formal requirements language, and (2) issues that relate to the broader requirements engineering context. The latter research issues assume the existence of a formal requirements language, but it is assumed that their investigation can be done fairly independently. Below, we treat all issues strongly dependent on the language as the "Formal Requirements Language" issue. All other issues are given their own summaries. The style of characterization is similar to that used in section 4.2.2.2. Finally, we identify candidate issues that were rejected for inclusion in the Formal Language Track.

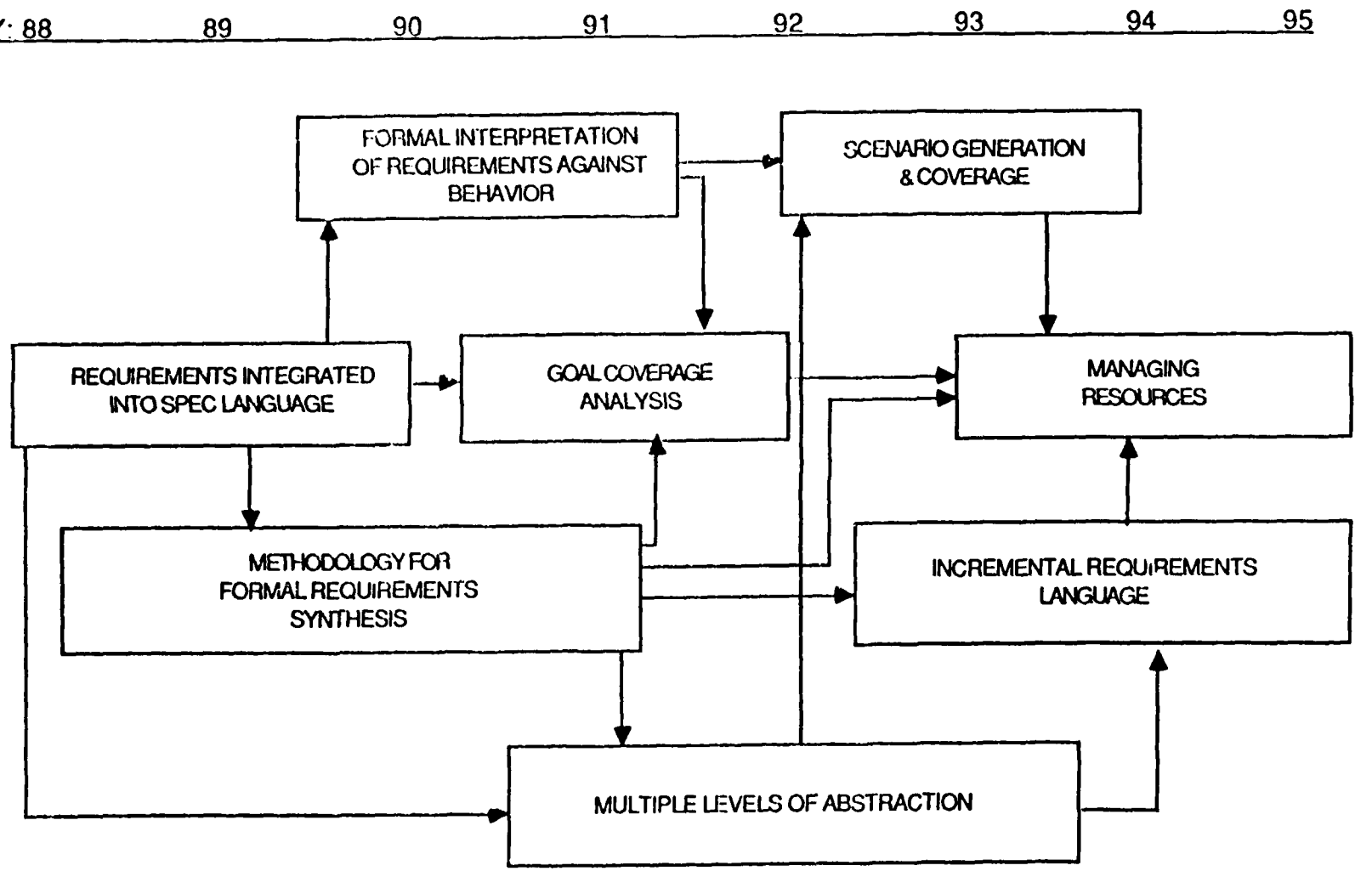


Figure 4.2.3.1-1 FORMAL LANGUAGE TRACK R&D ROADMAP

FORMAL REQUIREMENTS LANGUAGE

Where issue fits in process model: This issue will provide a common formal language for goals, requirements, and solution architectures. Thus analysis on each of these and synthesis/analysis between each of these can be given formal automated support. This would address most "transform" and "analysis" activities of the process model.

Research Objectives: Identified as individual research issues in figure 4.2.3.1-1: requirements integrated into specification language, formal interpretation of requirements against behavior, goal coverage analysis, multiple levels of abstraction, and an incremental requirements language.

Solution Approach: Expand existing formal specification language to include formal requirements statements. Share a common domain model and define requirements as predicates against behavior of specification. Formally execute specification to generate behavior against which to test requirements predicates. Include goals as requirements which can be further refined. Provide support for multiple levels of abstraction in stating requirements and specifications and mapping between them. Provide support for evolving requirements statements on basis of feedback from evaluation tools.

Recommendation: Funding priority: High. Risk: low for integrated language; high for reasoning and analysis tools. Formal language approach to requirements is highly recommended as a complement to the Evolutionary Track. It is higher risk and higher payoff and that payoff occurs later than in the Evolutionary Track. But it lays the foundation for earlier and more reliable detection of requirements problems and their use as a real design envelope.

METHODOLOGY FOR FORMAL REQUIREMENTS SYNTHESIS

Where issue fits in process model: Guides the synthesis of goals and requirements.

Research Objectives: Provide guidance for mission users and acquisition engineers in creating formal requirements.

Solution Approach: Extend structured specification methodologies to requirements and their formal expression. Extend the methodology to expression of goals and determining how to revise and refine them.

Recommendation: Funding priority: Medium-high. Risk: moderate to high. This research offers much promise in that it aids getting conceptualizations into formalisms, thereby making Formal

Language Track facilities accessible to a broader range of users, but success is uncertain.

SCENARIO GENERATION AND COVERAGE

Where issue fits in process model: In rapid prototyping: building scenarios that drive prototypes and by examining the generated behavior, determining whether the requirements are correct and complete. Requirements evaluation and reformulation: Issue also addresses support for determining whether the requirements are satisfied under a class of scenarios.

Research Objectives: Provide tool which determines whether requirements are satisfied by a specification with respect to a particular scenario. (The value of this check is, assuming satisfaction, any undesired behavior detected by the user during execution of the specification against the scenario will imply incorrect or incomplete requirements.) Expand this tool to handle classes of scenarios. Support automatic generation of a complete scenario from a mission user's outline of desired behavior.

Solution Approach: For generation of scenarios: reverse the flow of reasoning in symbolic evaluation to deduce the class of inputs to a specification that generates the behaviors desired by the user. For checking satisfaction of requirements against specification (prototype) behavior: use abstraction mapping/matching mechanisms to check whether the behavior the prototype generated is a legal instantiation (legal relative to the requirements).

Recommendation: Funding priority: High. Risk: near-term: low to moderate, but moderate to high for handling classes of scenarios. Automatic generation of scenarios and checking for requirements satisfaction are among the highest leverage capabilities of either track.

MANAGING RESOURCES

Where issue fits in process model: Supports all requirements engineering activities by managing the human and computing resources. Provides a basis for effectiveness measurements.

Research Objectives: Manage the human and computing resources needed to engineer a set of requirements, and track the resources through the engineering process.

Solution Approach: Formally describe all requirements engineering activities as tasks; identifying their dependencies, resources consumed, and results produced. Construct task manager which understands these descriptions and guides user in task selection. For multi-user efforts, expand task manager to coordinate all tasks and use of tools among users.

Recommendation: Funding priority: Medium. Risk: moderate to high. Success in this issue is strongly dependent on success in most other Formal Language Track issues.

Rejected Issues

The following issues were rejected for inclusion into the Formal Language Track. Reasons for the rejection are given.

REQUIREMENTS ANALYSIS METHODOLOGY

Rejected because existence of a formal language implies a basis for the formal treatment of descriptions, with "tools" automatically invoked in a data-driven problem-dependent manner. The "coordination" of such invocations is left to be addressed in the "Managing Resources" research issue.

SCENARIOS EXECUTION AND ANALYSIS OF RESULTS

Rejected as a separate effort because of its very strong dependence on the formal language. However, rapid prototyping will provide high leverage given the formal language approach; with additional value in the formal verification possible between the prototype and requirements (with respect to particular scenarios). This capability will be included in other Formal Language Track efforts.

NATURAL LANGUAGE

Considered as an aid to knowledge acquisition and formalization of descriptions. Rejected because that technology is already being extensively explored and its development is independent of our intended requirements use. As it matures, it will undoubtedly be incorporated into the RET. It was rejected as a RET-specific research area.

SOLUTION ARCHITECTURE SYNTHESIS

Rejected because that technology is being extensively explored and it examines an issue orthogonal to the primary focus here. However it will provide high leverage as it provides a basis for generation of rapid prototypes. As the technology matures, it will be evaluated for incorporation into the RET.

4.2.4 Resource Allocation

This section presents panel recommendations on the priorities of issues within each track and on the recommended allocation of R&D resources between the two tracks.

4.2.4.1 Prioritization in the Evolutionary Track

Research issues in the Evolutionary Track were prioritized in the order of preferred funding by the Requirements panel. Development issues were not prioritized relative to each other as they were all regarded as critical to the Tool Evaluation activity (see figure 4.2.2.1-3). The prioritization of research issues follows:

HIGH:

Scenarios Execution and Analysis of Results

Estimation of Cost, Risk, Time in System Development, Performance & Execution Costs Analysis

Scenario Generation Support & Scenario Coverage Analysis

MEDIUM-HIGH:

Domain Models and Information

Dynamic Analysis

Requirements Analysis Methodology

MEDIUM:

Requirements - Static Analysis

Requirements Evaluation

MEDIUM-LOW:

Goals and Requirements Synthesis

Testbed Effectiveness

LOW:

Validation of Prototype and Scenarios

Solution Architecture Synthesis

4.2.4.2 Prioritization in the Formal Language Track

Research issues in the Formal Language Track were prioritized in the order of preferred funding by the Requirements panel. As explained in section 4.2.3.2, all language-dependent issues were treated as a single issue, the "Formal Requirements Language" issue. The prioritization of research issues follows:

HIGH:

Formal Requirements Language

Scenario Generation and Coverage

MEDIUM-HIGH:

Methodology for Formal Requirements Synthesis

MEDIUM:

Managing Resources

4.2.4.3 Allocation of Resources to Both Tracks

To establish a relative allocation of R&D resources between the tracks, the panel partitioned the R&D issues into four groups and prioritized them. The groups were: Evolutionary Track research issues, Evolutionary Track development issues, the Formal Requirements Language issue, and Formal Language Track other issues (see section 4.2.3.2 for explanation).

The result of the allocation:

Group:	Funding allocation:
Evolutionary Track research issues	30%
Evolutionary Track development issues	29%
Formal Requirements Language issue	23%
Formal Language Track other issues	18%

To summarize, the panel suggests allocating funds between the Evolutionary Track and Formal Language Track on a 60%/40% basis. Within the Evolutionary Track, equal weight is given to the research and development issues. Within the Formal Language Track, the funds should be allocated between the language and other issues on a 60%/40% basis.

5.0 CONCLUSIONS

The panel recommends that RADC pursue a Requirements Engineering Testbed (RET) research and development program consisting of two tracks: an Evolutionary Track and a Formal Language Track. Their strategies are summarized below.

The Evolutionary Track

The "Evolutionary Track" proposes an evolutionary R&D effort to extend the current formalisms and tools. Initial efforts are toward the development of tools for prototyping interfaces and functionality, and in deriving performance estimates based on estimated or simulated work loads. Future efforts would develop tools and methods that aid in: (1) scenario development, analysis, and execution, (2) cost, risk, and performance analysis, (3) the acquisition, modeling, and usage of domain information, and (4) requirements analysis methodology. Future efforts will also go toward enhancing the prototyping tools and formalisms.

The Evolutionary Track also proposes several development efforts. (1) A database is needed to manage complex data such as requirements, prototypes, etc., and would feature powerful viewing mechanisms to simplify presentation of data. The database would also serve as the central repository for data and permit sharing data between tools. (2) A reconfigurable user interface is needed, and must provide uniform access to all tools and data objects. (3) Testbed tools must be tightly integrated, permitting their functionality to be manually invoked or invoked with changes in data (data-driven control). The integrated testbed must also track RET tool use to support later analysis of tool effectiveness, and guide the user in tool usage (e.g. methodologies).

The Formal Language Track

Currently, requirements, specifications and prototypes are partially (or completely) informal, and are separately and manually produced. This informality precludes tools which compare one level with the next and limits the types of analyses that can be done within a single level. The "Formal Language Track" attempts to eliminate these difficulties by creating a common formalism in which all three levels are expressed and in which the prototype can be generated automatically from the specification. Tools would be provided for analyzing the requirements and specifications and formally comparing them. The common formal basis for requirements and specifications would be used to automate the generation of scenarios to determine the satisfaction of requirements in a specification and the completeness and appropriateness of the requirements themselves.

Thus, the Formal Language Track proposes research effort be spent toward developing a single formal language for expression of

goals, requirements, and solution architectures (process model terminology). Additional effort would go toward researching relevant issues: (1) scenario generation and analysis, (2) synthesis methodologies (getting conceptualizations into formal descriptions), and (3) the management of resources (coordinating use of the resources supporting formal analysis).

Contrasting the Two Tracks

To contrast the two approaches, the formal language approach offers much more formal analysis to be done and thus has high payoff, but requires considerable research before it will be available. The evolutionary approach permits earlier RET hosting of capabilities such as: prototyping, scenario development and execution, and domain information collection and usage. The consequence is that Air Force users will be able to exercise these capabilities on their requirements problems before availability of the formal language. Also, the risk is distributed. These complementary risk/reward profiles strengthen the program and provide a natural phasing of capabilities.

Though not investigated in any detail, it is expected that some positive results developed in one track may impact or eliminate approaches being tried in the other, to take immediate advantage of the result.

To the extent that formalisms are adopted in the Evolutionary Track, the distinctions between the two tracks will be reduced or eliminated.

The R&D Program Time Line

RADC's currently-contracted tools will be hosted in the RET by early 1988. These tools will initially be loosely coupled, but by 1990, the objective is to tightly integrate them, and other new tools, through the RET database.

A few years later, capabilities from the Formal Language Track, beginning with a common requirements and specification language, will be available in the RET for test and evaluation. Thereafter, both tracks will be strengthening their tools and methods and using the RET as a test and evaluation vehicle.

6.0 EPILOGUE: RESULTS FROM A REVIEW OF THIS REPORT

Following completion of this report, copies were distributed for review. This section presents reviewers' comments and suggestions.

The review was carried out by computer scientists from both academia and industry. Their comments and suggestions were solicited on the first five sections of the report. The reviewers were:

Mack W. Alford - General Electric, Valley Forge, Pennsylvania,

MCC, two anonymous reviewers selected by Laszlo A. Belady - MCC, Austin, TX,

Harlan Mills - IBM and University of Maryland, Maryland,

Grula-Catalin Roman - Washington University, St. Louis, Missouri.

The comments below are reported anonymously. No comment should be taken as the unaminous opinion of all the reviewers, in fact there was no comment common to all reviewers.

Summarizing, the report was said to be technically strong, though with shortcomings.

Technical Strength

In general, the recommendations on the Requirements Engineering Testbed Research and Development Program were perceived as sound, going a good way toward addressing problems later encountered in systems development.

The Requirements Engineering Testbed concept as a vehicle for technology transfer and for maintaining research in systems definition was also considered to be sound.

The Shortcomings

In the minds of some reviewers, the most serious shortcomings were:

(1) The recommendations lean too heavily on formal specification techniques.

(2) The recommendations do not rest on a sufficiently adequate foundational basis, affecting the credibility of some of the recommendations on tools.

Responding to the first comment, it is the consensus of the panel that formal specifications will have a big payoff in the long

term, but relying on them in the short term is naive. That is why at least two-thirds of the 1990 Requirements Engineering Testbed is based on "informal" approaches, i.e. the Analyst and RPS. The role of formal specifications in the testbed is expected to gradually become more prominent after that.

Responding to the second comment, the panel agrees on the absence of a strong foundational basis, and knows of none. One major reason why long-term research and development programs are funded is to provide opportunities to identify such a foundational basis. One way to do this is to identify and develop promising interim approaches that give rise to new issues and ideas that can be explored. In the report, the panel attempted to define such a long-range program.

Additional Opportunities

All reviewers identified one or more research and development opportunities they felt should be investigated:

(1) Simple solutions such as enhancing the communication between mission users and system engineers, and using good people to do the systems engineering.

(2) What to do on ill-defined problems. If the problem can't be readily defined beforehand, all of the techniques explored by the panel provide little help.

(3) The "Requirements Volatility Problem". Mission and user needs change with time, thus the requirements change with time.

(4) A more flexible representation for scenarios and scenario sets. Benefits obtained would include a project-wide glossary, and the use of scenarios to explore the boundaries of system capabilities.

Responding to the first half of (1), and acknowledging (3), it should be recognized that the panel was investigating the requirements problem in the context of government procurement, which makes some solutions to these issues impractical.

Responding to the second half of (1), the panel agrees on the importance of using good people to do the systems engineering.

Responding to (2), the panel voiced no technique for exploring ill-defined problems other than prototyping, evaluating the prototype in near-actual use, and iterating. The panel feels that the requirements process model already addresses this, though implicitly.

Responding to (4), the report recommends that "Scenario Generation Support & Scenario Coverage Analysis" and "Scenario Execution and

Analysis of Results" research and development issues receive high priority for funding. In short, the panel agrees with the reviewer on the importance of scenarios, though maybe feels differently on where to place emphasis.

Miscellaneous Comments

One reviewer recommended that a critical objective of the Requirements Engineering Testbed should be to close the gap between "goals", formulated in terms of mission concepts, and "requirements", formulated in terms of system concepts.

The panel feels this to be a presentation issue. It feels the issue is addressed in the proposed program, and was one major reason for making the distinction between "goals" and "requirements" in the requirements engineering process model.

In addition there were a number of comments recommending further thought on parts of the process model: addressing performance requirements, design decomposition and representation, and the affect of size and complexity on the requirements problem.

A serious concern was expressed on the tool developments implied by the 1995 scenario. The reviewer felt that more thought was required on their justification.

The panel agrees on the conjectural aspects of the 1995 scenario. The panel was attempting to project feasible capabilities, as a means of identifying long-range objectives in the Formal Language Track of the research and development program. As knowledge is acquired through the Requirements Engineering Testbed program, the direction of anticipated tool developments should be revised.

Summary

Summarizing, the major concerns seem to be:

* A better foundation for representing requirements and techniques for eliciting and expressing them.

* Continued reevaluation of the formal specifications tool development program as to direction and justification.

* A more flexible contracting process, permitting a more opportunistic interleaving of definition, development, and evaluation efforts. Also, permitting better communication between mission users and systems engineers.

APPENDIX A: DETAILED CHARACTERIZATIONS OF TESTBED USERS

This section provides detailed characterizations of three classes of Testbed users: Mission users, Acquisition engineers, and Software developers.

Characterizations are given for three different times: "Now", "5 Years", and "10 Years". For each user class and each time, the following is described:

* BACKGROUND - degree of user sophistication with computer tools, languages, and applications
* ISSUES, QUESTIONS - what are the user's concerns with the requirements, what is his role?
* HELP - What kind of support would enable the user to satisfy his objectives?
* TECHNICAL ISSUES - What technical issues must be addressed if help is to be given to the user?

As RET R&D program focus is on helping the Mission user and Acquisition engineer, "Help" and "Technical issues" descriptions are not provided for the Software developer.

For each testbed user: "Issues, questions", "Help", or "Technical issues" characterizations that appear early in the time line generally persist throughout the remainder of the time line. To conserve space, we avoid repeating them.

Mission User.

Time: Now

background:

* Access to "time-shared" text-oriented terminals

* Computer literate, but no computer specializations

* (Typically) manages message handling systems : accessing/correlating info from local and remote databases, notifying others

* Limited networking:
  (1) user-to-user on mission-related job
  (2) transmittal of requirements documents among among acquisition engineers

Issues, questions:

* Will I be able to accomplish my mission?

* Will I survive under this scenario?

* Will I be "frozen out" because the computer system has removed control from me?

* Are accuracy and response time sufficient?

* Is the interface oriented to my needs?

* As the system wears down through attrition, will I be able to invoke appropriate back-up systems?

* What diagnostics are automatic?

Time: 5 years

background:

* Graphical interface (vs. textual): forms/icons style of computer dialog

* One PC per desk

* Networking standard among:
  (1) user-to-user with respect to a mission-related job;
  (2) (RADC) acquisition engineers, sending back and forth requirements documents

* Strong resistance to formal methods (prefers learning time of 1 hour)

Issues, questions:

* As the system wears down through attrition, will it degrade "gracefully"?

* What repairs are automatic?

* Familiar with some canned programs

* Prefers interface suited to his application domain

* Training: application-oriented

Time: 10 Years

background:

* Networking: potential for conversationamong all three users (especially mission user and acquisition engineer) (link to software developer indirect through agent)

* Electronic mail paradigm/culture

* There will be a mission-associated language, with which user will be familiar

* Experience with knowledge-based systems advice-giving, fourth generation languages (resulting in automatic generation of programs)

Issues, questions:

* Is there a latent system error that will cause mission failure?

Acquisition Engineer.

Time: Now

background:

* Shared dumb terminal

* Shared PCs just beginning

* Knowledgeable of application

* Computer literate and one or more specializations

* Familiar with a word processor and spreadsheets

Issues, questions:

(6 questions relating to whether the requirements are right:)

* Do they fulfill the user's needs?

* Can the sytem be built?

* Are cost/schedule expectations realistic?

* Very limited office automation (10%)

* Are mission user concerns/needs understood by the developer?

* Is the mission user sensitive to development constraints?

* Is each requirement quantifiably demonstrable?

* Can procurement selection be justified on technical grounds?

* Is competition being fostered (will there be many responses to my RFP)?

Time: 5 Years

background:

* One PC per desk

* Limited networking:
  (1) with mission user,
  (2) possible objective: tie developer in for big development programs

* Limited office automation (20%)

* Increased specializations

* Limited prototyping for requirements evaluation and validation (associated with incremental development, especially interfaces)

* Metrics for evaluating alternatives

issues, questions:

* Which requirements should be evaluated?

* For certain requirements, are there alternatives? What costs/benefits do they entail?

Time: 10 Years

background:

* LANs, providing access to a wide spectrum of computer resources

* Limited decision tools (little AI)

* Breadth of computer knowledge

issues, questions:

* Can software procurements go fixed price, warrantied?

* Office automation (50%)

* AI-based decision tools

* wide use of prototyping for requirements evaluation and validation

Software Developer.

Time: Now

background:

* One language fluency

* Workstations on LANs

* Access to large machine

* Office automation support

* No automated allocation to multiple processors

* Large group of helpful peers

issues, questions:

* Do requirements over-specify a design? Or, are requirements too mission-oriented (high-level)?

* Does the development staff have the required skills?

* What are the hard real-time problems (which functions in which scenarios produce bottlenecks)?

* Are cost/schedule constraints compatible with function/performance requirements?

Time: 5 Years

background:

* Highly supportive workstation

* Fluency with several workstation environments

* One physical workstation per desk, on networks

* Comfortable in using a number of languages

* No design/architecture support tools

* Development host and operational host separation

issues, questions:

* Can I eke out a competitive edge through systematic experimentation of system alternatives?

* Does my development team have access to and knowledge of the best development tools and techniques to do the job?

* Functions parsed into multiple processes

* Breadboarding of all critical design ideas

Time: 10 Years

background:

* Whole environment fosters "competitive differentiation"

* Prototyping testbed for evaluating real-time system requirements

* Design/architecture concept support tools

* Expert system technology matures

* Build software first, then order hardware highly tailored to software architecture

Issues, questions:

* How can I demonstrate, or prove, a priori, error-free operation?

Now a characterization of the kinds of support that would enable the mission user to attain his objectives.

Mission User.

Time: Now

help:

* Expert committee pulled together by mission user
* Limited incremental development (pre-planned program improvement) for putting together large systems

technical issues:

(partial solutions already exist)

Time: 5 Years

help:

* Conceptual model (mission-oriented, stated in English) of what is needed (basis of statement of need)
* Limited use of operational models of major subsystems; providing opportunities to stress-test critical functions
* Some isolated expert systems based upon domain models

technical issues:

* Application-oriented VHLL for prototyping and simulation
* Advanced data base concepts
* Real time: user experiences unfolding battle
* Reusability
* To support real-time simulation: high-speed processors
* Risk evaluation via rapid prototyping (seeding errors at requirements level)
* C3I rapidly reconfigurable environmental simulator (hardware and VHLL)
* Parallel algorithms to provide opportunities for concurrent processing

Time: 10 Years

help:

* Operational models widely used
* Partially integrated domain models

technical issues:

* Artificial Intelligence:
  ** Rule generator for VHLL
  ** Rule critiquer
  ** domain-model support

A characterization of the kinds of support that would enable the acquisition engineer to attain his objectives.

Acquisition Engineer.

Time: Now

help:

* Very limited rapid prototyping
* Incremental development routinely done
* Simulation models
* Scenario generation (via graphics, etc.)
* Cost estimation models
* Proposal evaluation: architecture evaluation especially (i.e. high level design) (mental exercise); program development plan (mental exercise)
* Formal review process (no separation of views), and walkthroughs
* Experimental use of new languages, tools, methodologies (especially in the validation of the requirements delivered by the contractor)
* Performance models (assessments of computer configuration (through queueing models))

technical issues:

(partial solutions already exist)

Time: 5 Years

help:

* Limited use of rapid prototyping for evaluation and validation of requirements
* risk/cost evaluation of alternatives (for requirements specification and evaluation of design approaches in proposals)
* Static analysis of requirements:
  ** syntax-checking
  ** ad-hoc measures for evaluating
* Ad-hoc measures of evaluating the tools that produced the requirements
* Viewpoint enactment tools in limited use (separation of views)

technical issues:

* common VHLL/database for exercising requirements with many tools
* VHLL for rapid prototyping
* advanced data base concepts for knowledge representation (domain modelling) and update dependencies
* distributed environment
* test case generator and scenario generator for stress-testing
* risk/cost models/metrics
* domain models (functionality, performance, test data characterization)
* (other issues similar to mission user)

Time: 10 Years

help:

* Fully integrated models:
  ** domains
  ** simulation
  ** system definition
  ** scenario
  ** validation
* Significant use of rapid prototyping for evaluation and validation of requirements
* Increasingly formal and semantical static analysis

technical issues:

* Artificial intelligence:
  ** mission --> plans representation
  ** modelling support
  ** update dependencies support
* Advisor on procurement strategies

Time: 15 Years

help:

* Requirements specification standards: Each requirement is a quantifiably measurable augmentation or constraint to system behavior (through testing or IEEE standard evaluation)

APPENDIX B: DOD STANDARD 2167 AND THE REQUIREMENTS ENGINEERING TESTBED PROCESS MODEL

DoD-Std-2167 prescribes requirements for the development and acquisition of software in terms of six software development life cycle phases: software requirements analysis, preliminary design, detailed design, coding and unit testing, integrating and testing of Computer Software Components (CSCs are aggregates of software units) and testing of Computer Software Configuration Items (CSCIs are the highest level aggregation of CSCs and units). Its emphasis is primarily managerial. It concentrates on the planning and scheduling of products and reviews by which a software development and acquisition can be controlled. Technical guidance on how the software should actually be designed and built is minimal. It is implied by the hierarchical structuring of the CSCI, CSC and software units to be a top down decomposition based on functional requirements, data flow requirements or other design considerations. The software development model prescribed by 2167 is also directed for use in the development of software in all stages of the system life cycle. This includes concept exploration, demonstration and validation, full scale development and production and deployment.

The Requirements Engineering Testbed (RET) process model is a partially iterative set of transformations to a set of descriptions which take user needs from a very informal, wish list stage, to formal requirements and partial designs. Its primary intent is in describing the technical nature of the products and the transformations which they undergo. It does not explicitly address management and acquisition issues. It is intended to iteratively evolve user needs into requirements during the software development activity which precedes implementation. There is nothing inherent in the model which would preclude its use during post implementation activities, for example, to incorporate changes to user needs and their related requirements.

The purpose of this appendix is to assess the relationship between 2167 and the RET process model. All of the Software Requirements Analysis phase activities, products, reviews and developmental baselines defined by 2167 will be identified and their relationship to the RET model will be explored in terms of the kind of information which the RET model produces and which would be usable within the requirements of 2167.

Software Requirements Analysis Activities defined by 2167

> Establish control of various management documents, such as, the Software Development Plan (SDP), Software Standards and Procedures Manual (SSPM), Software Configuration Management Plan (SCMP) and Software Quality Evaluation Plan (SQEP).

None of these plans or manuals are addressed by the RET model.

Analyze the previously developed developed, preliminary Operational Concept Document (OCD) and describe the system mission, its environment and the functions of the system's computer system.

The OCD is a description of how the target system will appear to its users and the ways that the users will interact with it. The OCD provides a statement of the role or mission of the target system. It describes the physical/tactical environment in which the target system will be used and identifies its performance parameters. It lists and describes the functional roles in which the target system will support its users. It describes the capabilities which it will provide to these users in these roles. It outlines the hardware (including computer) complex which will be required to support the target system. Finally, it gives the plans for maintaining, upgrading and adapting (to changing requirements) the target system.

The RET process model recognizes that good user understanding of the mission, application and specific job actions is critical in performing the RET process model transformations which take wishes to goals and goals to requirements. This body of knowledge currently resides with the user. The RET model recognizes the need to formally represent it in domain models. This information could be used to help directly in the analysis and description activities which 2167 requires.

Analyze for adequacy, testability, understandability, validity and completeness the previously developed preliminary versions of technical specifications, such as the System/Segment Specification (SSS), System Requirements Specification (SRS) and Interface Requirements Specification (IRS).

In the context of 2167 this activity and its predecessor, i.e., analyze OCD, form a logical sequence of events leading up to the (next activity) definition of requirements. The RET process model does not explicitly incorporate the analysis of "outside" requirements or specifications. It assumes that all such information will have originated within the RET model, will be expressed in a native RET representation form and will be analyzed using RET model facilities.

Define a complete set of functional, performance, interface and qualification requirements for each CSCI including programming constraints and standards, design constraints and standards, adaptation, quality factors and preparation for delivery.

The RET process model is directly responsive to most of the requirements of this 2167 activity. Completeness is, of course, elusive. Given the RET model's static analysis, animation and prototyping capabilities, it should be possible to produce a set of requirements which are as complete as current understanding of the system and its environment allows.

Function and interface requirements are addressed by the model's language for representing requirements and the methodology for its usage. Performance requirements are addressed by the model's language elements which permit the representation of goals.

Qualification requirements refer to tests which can be used to validate the requirements. Although the RET model provides validation mechanisms through prototyping, it does not explicitly structure test sequences (inputs, expected outputs, coverage criteria, etc.) which are designed to demonstrate requirements compliance during final acceptance testing.

Since the RET model deals with candidate solution architectures, it addresses design constraints at this preliminary level. It does not address programming constraints or standards for either design or programming.

Adaptation to changing requirements is implicit in the requirements language supported by the model and explicit in the model's iterative, refinement nature.

The quality of products produced by the RET process model, although not explicitly captured in the model, is a research issue to be addressed during the long term construction of the RET.

Identify structured requirements analysis tools and techniques.

Implicit in the RET process model.

Conduct internal in-process reviews of requirements documents for the purpose of improving the quality of requirements prior to delivery to the customer.

The RET model incorporates an explicit evaluation step as part of its iterative process of requirements refinement. This process includes dynamic analysis of requirements and analysis (performance, cost, risk) and prototyping of partial designs.

Software Requirements Analysis Products defined by 2167

Updated versions of the SDP, SSPM, SCMP and the SQEP.

The RET process model does not address these management documents.

Updated version of the OCD.

The domain models, which are major elements in early RET process model transformations, could provide valuable mission, application and user cognitive information during the OCD updating process.

Reports of internal reviews for recording and summary purposes.

The RET process model does not address management reporting issues.

System Requirements Specification and Interface Requirements Specification for each Computer Software Configuration Item.

These documents could benefit directly from the requirements and partial designs which are the major products of the RET process model. However, several format and content differences must first be reconciled in each case:

SRS

The SRS requires documentation of design and implementation information such as programming requirements (language, compiler/assembler, programming standards), design requirements (sizing, timing, standards, constraints) and interface requirements. The RET model clearly does not address programming issues. It may be able to assist with some design issues which are likely to surface during preliminary design. Candidates include performance constraints, interfaces between major software components (CSCIs) and interfaces between major software components and major hardware components.

The actual functional and performance requirements certainly could be directly found in the requirements and partial designs produced by the RET process model. However, format is a potential problem because the SRS demands that these descriptions be given as a functional hierarchy with each function being described by its inputs, processing and outputs. The same requirements are also placed on all subfunctions.

Although the exact representational form of the RET process model's requirements has yet to be determined, it is likely that is underlying model will be object oriented. In this case some translation between the objects, relationships and operations of the RET model and the SRS functional breakdown will be required. This appears to be more than a minimal effort, since it is likely to require human interpretation.

SRS requires low-level operational information such as site dependent environment data (e.g., radar ranges), system parameters which vary according to operational needs (e.g., allowable trajectory deviations) and system capacities which are likely to change (e.g., secondary storage). The RET model is not intended to expose information at this level of detail.

The SRS includes the description of 12 quality factors: reliability, correctness, efficiency, integrity, usability, maintainability, testability, flexibility, portability, reusability and interoperability. The RET process model deals explicitly with only reliability. Correctness (consistency and completeness checking) and usability (interface prototyping) are addressed to some extent.

The SRS includes support items such as facilities, equipment, software, personnel and training which would be used during the development, operation and support of software associated with the requirements being documented. The RET process model does not address these issues.

The SRS requires qualification of the software which is generated from the stated requirements i.e., that it actually satisfies the requirements. Qualification methods include inspection, demonstration (observable functional operation), testing (collection and subsequent examination of data) and analysis (processing of accumulated data). The RET process model supports all of these methods to some degree.

Inspection and demonstration of functional operation can be done through rapid prototyping of both interface and functional requirements. The RET model provides for performance and reliability analysis of partial designs. Results of these analyses could be helpful in supplementing the testing and analysis qualification methods.

IRS

The Interface Requirements Specification describes the requirements for one or more interfaces between major software components (CSCIs) and other configuration items (software and hardware) or critical items in the system. The IRS requirements for describing the interfaces and their qualification are identical to those contained in the SRS. The IRS is meant to be an elaboration of the SRS with respect to component interfaces. Consequently, the comments given above for these aspects of the SRS are also applicable here.

Software Requirements Analysis Reviews defined by 2167

2167 requires a Software Requirements Review (SRR) at the completion of the software requirements analysis phase to demonstrate the adequacy of the OCD, SRS and IRS.

The RET model defines a requirements evaluation activity as part of its iterative process for refining requirements and partial designs. However, the intent of the RET model is to perform evaluation as an integral part of the requirements refinements process. This differs from 2167's SRR which is a final, formal review, intended to convince management that the requirements are acceptable for use in the next (preliminary design) phase of the software development cycle. In the sense that the SRR acknowledges requirement problems and accommodates their resolution, it resembles the RET model's evaluating activity. The RET model's evaluation activity could be used as a single review point, but this would have to be considered a degenerate case of RET model application.

Software Requirements Analysis Baselines defined by 2167

2167 defines the SRS and IRS to be allocated baseline for further design and implementation activities at the time these documents are accepted by management, i.e., at the successful conclusion of the SRR.

The RET process model does not address allocation or configuration of project baselines.

Conclusions

The relationships between DOD-STD-2167 and the RET process model are many and complex. This results from 2167's being a demanding standard for software documentation. Several more specific conclusions are possible:

The RET process model clearly does not satisfy all the documentation requirements of 2167. As would be expected, its major deficiencies are related to 2167's management and acquisition requirements, specifically the various plans for organizing, procuring, configuring, baselining and assuring the quality of a large scale software development in accordance with established standards and procedures.

The requirements and partial designs produced by the RET process model can provide direct assistance in preparing much of the technical information required by 2167, such as the functional, performance, interface and qualification requirements.

The RET process model's mission, application and user cognitive domain models can be used to generate portions of the Operational Concept Document.

The RET process model cannot assist in the analysis of previously developed requirement and specification documents, such as the System/Segment Specification, as required by 2167. The RET model does not recognize the context (earlier software development cycle phases) in which such documents were developed, nor are such documents in the language and format required for analysis by RET processes.

The requirements evaluation activity of the RET process model is iterative in nature and does not map well into the single review point mechanism required by 2167.

The RET process model does not address certain low level design and implementation information, such as language and compiler specification, which 2167 requires.

The RET process model will most likely be based on a paradigm (object oriented) which will not be isomorphic to the hierarchical structuring of software elements as required by 2167.

The RET process model does not address a number of software quality factors, such as correctness, integrity and maintainability, which 2167 requires.

APPENDIX C: PROCESS MODEL: DETAILED CHARACTERIZATION

C.1 Definitions

In figure 3.1-1 of section 3, boxes, clouds, and numbered arcs represent information relevant to the engineering of requirements. In this subsection, we give detailed characterizations of each information type: what it expresses, its role in requirements engineering, its properties, and in what notations it is expressed. We consider the information types in the order we might encounter them looking down figure 3.1-1.

ENGINEERING CONTEXT DESCRIPTIONS

Represented as a cloud because of its external and meta-level status.

Express. There are external constraints and concerns influencing requirements engineering. There will be limitations to the amount of effort that can be spent ("effort constraints" in the figure). Fully engineering a "final requirements and partial solution architecture" is only an ideal, so it is important to keep a pricritized list of objectives in hand ("goals of requirements engineering exercise" in the figure).

Role in requirements engineering. Focusing in on the RET context, the above concerns are more acute. A principal goal of the RET is the evaluation of requirements engineering tools and techniques (i.e. measurement of process and product). This evaluation requires many RET experiments, each of limited effort and carefully selected objectives. Hence the increased acuteness. Concerns of making a limited resource (the RET) effective to a broad audience (Air Force requirements engineers) will also influence the nature of the requirements engineering efforts ("testbed effectiveness" in the figure).

WISH LISTS

Represented as a cloud because of its external and undocumented status.

Express. Wish lists are expressions of desired attainments, but unlike goals, there is generally no attempt at documenting the context of the desire. Wishes are generally characterized as "likes" and "don't likes" in the domain-specific terms of existing solutions. Wishes originate with mission users and administrators in the field.

Role in Requirements Engineering. Wishes motivate the need for a new system; they are an external source in the process model. They are not formally maintained unless they can be precisely stated in goals.

Properties. Very informal, often poorly conceptualized. Wishes may be inconsistent, even if from the same source. Worse yet, they may be misleading; a "like/dislike" statement is likely to be imprecise and therefore ambiguous. Wishes may change.

Notations. Wishes originate in people's heads. Expressed in natural language. Domain-specific terminology will likely be employed.

GOALS

Express. Goals express desired attainments, but goal expressions should characterize the context of the desire too. Target system users, administrators, and procurers all have goals, so there may be many sources for goals. Normally, each set of goals characterizes the viewpoint of a single role in the operation, administration, or maintenance of the target system. Such a characterization includes: mission performed, interactions with target system and other users, and desired attainments (e.g. "immediate response") expressed as constraints on the behavior of the target system (perhaps thru a scenario).

Goals can be "guiding philosophies", documenting the metrics of concern (e.g. "a fast tank"). In addition to characterizing desired system behavior, they can characterize desired cost or desired availability.

Role in Requirements Engineering. Goals: (1) Help control complexity early, specifically requirements collection and validation, because such efforts are naturally partitioned into different viewpoints, as are goals. (2) Goal expressions document the source and context of expectations on target system behavior, which sometimes must be explored during reviews or during maintenance.

Properties. Within a set of goals, consistency should be maintained, but between sets of goals, inconsistencies and conflicts will be common, reflecting the different missions in which the target system must participate. Concerns of technical achievability are second-order.

Notations. Currently, goals are rarely expressed, and then only in descriptive English prose. We envision use of domain/mission-specific formalisms and general requirements notations.

REQUIREMENTS

Express. Requirements characterize the target system in terms of: its operational environment, its interfaces, its functionality, and constraints on its performance, reliability, development and availability. Requirements express what is needed, but also

document assumptions on how external agents (users, existing systems) will interact with the target system.

Role in requirements engineering. Obvious.

Properties. Requirements should be operationally testable and consistent. The requirements should constitute a closed (complete) external model of the target system and its environment. Requirements should reflect what is technically achievable, but the developer has full implementation freedom within the limits established by the requirements.

Notations. Requirements have generally been informal and stated in a natural language, but future requirements specifications will utilize more formal notations as complexity of the needs to be conveyed increases.

Notational mechanisms are often employed that factor requirements into different concerns and/or levels of description.

SCENARIOS

In the process model, a scenario is part of a goals or requirements description.

Express. Scenarios characterize hypothetical interactions that emphasize target system role(s).

Typically, a scenario description has three kinds of parts: (1) a characterization of the initial state before the interaction, (2) a sequence of actions against the system, and (3) a characterization of the expected results (or final state). Scenarios may consists of several action sequence and expected result pairs.

Role in requirements engineering. Scenarios can aid the communication of goals and requirements by illustrating expected usage. Or scenarios can illustrate expected system behavior in response to exceptional or stressful input, thereby having the role of a system constraint. Scenarios can be used as test data, especially in exercising prototypes and in system acceptance tests.

Properties. Being part of a goals or requirements description, scenarios will tend to exhibit their properties in roughly similar degree, e.g. in technical achievability and operational testability. Scenarios should be consistent with the description of which they are a part.

Scenarios may be fine-grained in action/result characterization or coarse-grained.

Notations. Natural language and domain-specific terminology are employed. Some formalization of their structure is not uncommon; for example, representing the action sequence by a command file (sequence) of procedure calls.

SOLUTION ARCHITECTURES

Express. Solution architectures characterize how the target system can satisfy the requirements. Solution architectures are descriptions of the target system as a composite of parts (e.g. objects, functions) and resources (e.g. people, software, hardware) and how the different parts utilize the different resources.

Role in requirements engineering. Aids evaluation of requirements by: (1) aiding understanding feasibility and development implications (thru cost and risk analysis), (2) providing a basis for prototyping functionality, interfaces, and performance which by means of exercises and simulations lead to a better understanding of user needs, and (3) aiding making trade-off decisions.

Properties. Solution architectures need not be complete "system parts and resources" characterizations. They must be internally consistent and should be consistent with the goals or requirements they were constructed to stress.

Notations. Solution architectures are mostly formal descriptions. Design notations may be utilized for expressing the partitioning of the system into parts, and algorithms for expressing resource utilization strategies.

As with requirements, notational mechanisms are often employed that structure the descriptions into partitions and levels.

DOMAIN MODELS

Generally, domain knowledge is (cohesive) knowledge that persists beyond a single or short-term application. Thus if collected, such knowledge can be reused and referenced by goals, requirements, and solution architectures of several projects. This is the reason for its separate yet special status (numbered arcs) in the process model figure.

Express. A domain model is an encoding of knowledge specific to a domain. The knowledge is characterized thru a mixture of terminology, basic facts, and empirically-derived relationships. Domain expressions can be as rich and complex as expressions of knowledge, in general. Knowledge within a domain may exist at different levels, e.g. knowledge of how to use other domain knowledge (i.e. meta knowledge).

Role in requirements engineering. Domain information has value throughout requirements engineering.

Examples identified in the process model (not a complete or mutually exclusive set): (1) Mission models - the goals and work style of a particular mission user (e.g. intelligence analyst). Aids in creating goals descriptions. (2) Application domain models - the operations available in a particular application (e.g. text editing). Aids in understanding user likes/dislikes and in defining systems that perform similar applications. (3) User cognitive models - how much will human capabilities (such as decision making) be stressed. Aids in creating and analyzing requirements descriptions. (4) Software resources (e.g. operating system, DBMS), algorithms (e.g. search, sort), and (5) hardware models (e.g. networks, run-time environments, and performance attributes). Both aid in designing solution architectures and in their analysis.

Properties. As noted, domain information persists. A domain model should be conceptually cohesive in the sense of constituent parts relating to the same thing. The base terminology ("axioms") of a domain may be selected with different stability and fidelity in mind.

Notations. Mostly informal and natural language. As knowledge representation and acquisition technology improves, use of formalisms in expression of domain information will increase.

FINAL REQUIREMENTS AND PARTIAL SOLUTION ARCHITECTURE

The final set of requirements and architecture ideally are the result of many compromises and factor in many concerns. They are the end product of the requirements engineering process.

Properties. As complete, consistent, feasible, operationally testable, and traceable to user needs as the technology and time allow.

C.2 Process Model Activities and Support from 1990/1995 Tools

In figure 3.1-1 of section 3, circles, ovals, and rounded-corner boxes represent the activities relevant to the engineering of requirements. In this subsection, we give detailed characterizations of these activities in terms of: termination/invocation criteria, and tools that will assist the activity in the 1990 and 1995 RET. These tools are products of the RET R&D program and are described in greater detail in section 4 and appendix E. The 1990 tool-support characterization is made in terms of the Evolutionary track. The 1995 characterization is made in terms of both the Evolutionary and Formal Language tracks. We consider the activities in the order we might encounter them looking down figure 3.1-1.

Goals Synthesis from Wish Lists

Through user interviews/reviews and reference to appropriate domain models (e.g. mission models), wishes are refined and documented with appropriate context, creating goals.

Termination criteria: Terminate when a complete and consistent set of goals has been constructed for each viewpoint.

Tool support in 1990 RET: DBMS-based mechanisms to support reuse, syntax-directed editing, and view management. The Analyst might be enhanced to utilize these mechanisms.

Tool support in 1995 RET: Evolutionary Track: Advanced DBMS to support access and manipulation of domain models by both users and other tools and to help state goals. Tools that support: knowledge acquisition, domain-specific dialogues, and expert critiquing.

Formal Language Track: Nothing beyond the Evolutionary Track approach is planned.

Requirements Synthesis from Goals

Conflicting descriptions in the sets of goals must be reconciled, creating a consistent model of the operational environment of the target system. Expectations of target system behavior are transformed into precise functional requirements. Scenario construction and analysis may aid stating the nonfunctional requirements, such as performance and reliability.

Termination criteria: The requirements should be complete (relative to the goals and scenarios considered) and consistent. On termination, the requirements should be traceable to the goals.

Tool support in 1990 RET: The Analyst tool enhanced with DBMS-based mechanisms (see "Goals Synthesis from Wish Lists").

Tool support in 1995 RET: Evolutionary Track: (See "Goals Synthesis from Wish Lists".)

Formal Language Track: because goals and requirements are stated in the same formal language, no language change is involved in going from goals to requirements. Instead, there is selection and refinement of goal statements via a methodology for formal requirements synthesis.

Static Analysis on Goals and Requirements

Static analysis can reveal some types of inconsistencies and incompleteness. If goodness meta-models (e.g. user cognitive

models, models of safeness) exist, they can be employed to derive properties (e.g. quality of cognitive support, degree of safeness).

When invoked: The principle to be followed is that any kind of analysis that can be done automatically should be done as early as possible. Whether to do a manual analysis requires consideration of the implied value added and effort required.

Tool support in 1990 RET: The Analyst supports some consistency and completeness checking. Almost all other static analysis on goals and requirements will be manual.

Tool support in 1995 RET: Evolutionary Track: Consistency and completeness checkers based on: (1) a controlled natural language representation of requirements, and (2) domain knowledge. Predictive and quality metrics and tools for quality assessment and critiquing.

Formal Language Track: The use of a formal language implies that statements are formally analyzable and that proof techniques can be used on the underlying predicates to determine things that are subsumed and incompatibilities.

Dynamic Analysis

Dynamic analysis helps one to understand how completely and correctly the requirements address a stated concern (e.g. a reliable interface to a set of sensors, or operator training). Relevant requirements are selected and a walk-through is conducted. Appropriate tool support and animation would be very helpful.

When invoked: When examining requirements for goal and wish coverage or for considering spontaneous or new concerns, such analysis is appropriate.

Tool support in 1990 RET: The Analyst tool will support an automated walk-through of requirements (i.e. an analysis of dataflows through the various user viewpoints which interact with the target system) and animation of requirements (i.e. direct execution of user or system actions given an initial state and some inputs).

Tool support in 1995 RET: Evolutionary Track: An animation-based dynamic browsing capability: (1) for analyzing requirements against domain knowledge, and (2) for analyzing requirements interrelationships.

Formal Language Track: Nothing beyond the Evolutionary Track approach is planned.

Candidate Solution Architecture Synthesis

There are different ways to partition the target system into parts. Different resources can be selected, and different resource utilization strategies can be used to control access to the resources. Thus the principal activity is making design decisions.

Termination criteria: There are different reasons for synthesizing solution architectures. These lead to different termination criteria. If the reason is: (1) Prototyping - we need enough architecture to produce the behavior to be investigated. (2) Estimating performance, reliability, or development cost/risk - we need enough architecture to make the relevant assessment. (3) Identification and definition of a satisfactory solution approach - we need to define an architecture which is sufficiently complete to determine with high confidence that the requirements can be satisfied.

Tool support in 1990 RET: The VHLL Prototyping tool and Rapid Prototyping System tools will provide modest design capabilities such as support for functional decomposition and allocation, concurrent processing, and characterization of computer systems and communication networks.

Tool support in 1995 RET: Evolutionary Track: Minimal, although use of predictive metrics with requirements might reduce need to do solution synthesis.

Formal Language Track: It is possible that there will be a capability to automatically suggest candidate solutions, but this is a very speculative area, and is not a focus.

Rapid Prototyping

After defining a solution architecture that produces or simulates selected behaviors of interest, a prototype is derived and exercised through a mixture of user-controlled and canned scenarios. Thereby, an evaluation of the selected behaviors is elicited.

Termination criteria: The prototype must be sufficiently complete for exercising. The exercising must be sufficiently thorough to either validate the selected behaviors or provide a good idea of what are the alternative candidates.

Tool support in 1990 RET: For 1988: Rapid Prototyping System tools (RPS) for interpreting prototypes against canned scenarios; the VHLL System Prototyping tools (VHLL) for executing prototypes that interact with users or simulations; and the Analyst tool for scenario-based symbolic performance estimates. To prototype human interfaces: especially RPS, but VHLL too. To prototype functionality: VHLL. To prototype for performance: all tools.

For analyzing results of prototype execution against a scenario: each tool provides a specialized analysis capability. A database will hold results and data management facilities will aid human analysis.

Around 1990, these prototyping capabilities may be enhanced with animation and with dynamic probes to determine coverage. There will be a methodology, possibly metric-guided, possibly based on CORE, guiding when to prototype.

Shortly thereafter, a knowledge-based simulation system will aid building adaptive scenarios that interact with prototypes.

Tool support in 1995 RET: Evolutionary Track: Knowledge-based aids for evaluation of prototype sensitivity complemented with: metrics-supported analysis and a methodology for when to prototype. A static coverage tool that analyzes the prototype, determining what system parts are exercised by the scenario.

Formal Language Track: (1) The solution architecture is the prototype. When driven by a scenario, the prototype generates behavior which is automatically checked for satisfaction of requirements. The payoff from the checked prototype is this: user-detected undesired prototype behavior implies incorrect or incomplete requirements. This helps the user carry out his role of checking the requirements. (2) Machine-assisted, user-guided generation of scenarios.

Analysis: Performance, Reliability, Development Cost, Risk

A candidate solution architecture is subjected to analysis, both manual and automatic, to determine its operational goodness (e.g. implied performance and reliability) and development goodness (e.g. implied cost and risk).

When invoked: When a candidate architecture is sufficiently complete to produce a high-confidence analysis.

Tool support in 1990 RET: The Rapid Prototyping System aids construction of usage-sensitive performance models.

Tool support in 1995 RET: Evolutionary Track: Metrics and tools for estimating cost, time, and risk from solutions, even if solutions have a distributed or knowledge-based architecture. Tools for fault-handling and reliability analysis.

Formal Language Track: The use of a formal language provides a formal basis for deriving estimates of what the operational performance, etc., will be. Otherwise, does not offer much beyond the Evolutionary Track.

Requirements Evaluation and Reformulation

Performing dynamic analysis, rapid prototyping, and analysis of operation/development implications leads to insights on requirements correctness and alternatives. These insights suggest a new/alternative version of the requirements. Such a version is constructed.

When Invoked: After sufficient analysis of the current version makes clear the need for a new/alternative version.

Tool support in 1990 RET: Various analysis tools and a CORE-based methodology.

Tool support in 1995 RET: Evolutionary track: Various analysis tools and a methodology, possibly metric-guided, combining predictive metrics with prototyping.

Formal Language Track: support for determining whether the requirements are satisfied under all possible conditions (scenarios driving the solution architecture). Such support will be based on a statistical and symbolic analysis of the behavior space.

Exit Testbed

In the RET there will be limitations to the amount of effort that can be expended in a particular requirements engineering exercise. There will be a list of specific objectives to be attempted in the exercise. These are documented in the Engineering Context Descriptions, which controls the duration of the exercise.

Presumably there are some post-exercise activities: (1) Documenting the objectives, constraints, and results. (2) Evaluation of tools and techniques used: documenting each tool and technique, the context of its use, problems encountered, and the character of success.

Tool support in 1990/1995 RET: See "Measurement of Process and Product" below.

Measurement of Process and Product

This activity does not explicitly appear in figure 3.1-1. As in the case of the "Engineering Context" cloud, it is a meta-level activity. As a key objective of the RET is the evaluation of tools and techniques, this activity is included here.

RET activities must be carefully planned, metered, and measured to assist making an assessment of relative tool/method strengths. These assessments are then used to guide development of new tools and techniques.

Tool support in 1990 RET: An instrumented testbed of integrated tools will support relative effectiveness measurements. Prototyping results will provide basis for making such measurements. A testbed methodology will direct RET experimentation.

Tool support in 1995 RET: Evolutionary track: Metrics will provide an improved empirical basis for determining effectiveness.

Formal Language Track: as more of the requirements engineering process will be formalized, it will be easier to develop appropriate measures of the leverage provided.

C.3 Expression of Information in the 1990 RET

In this subsection we discuss the kinds of information that 1990 formalisms will help to express.

Problem Domain Information for Deriving Goals & Requirements

We call information employed in deriving Goals and early Requirements "problem domain information". We give below a general characterization of what problem domain information is formally capturable in 1990.

Extrapolating on envisioned 1987 Analyst formalisms and functionality, the following problem domain information can be semiformally expressed:

Problem Domain information:	Notation for expressing:
* For each agent/role:	
Role description	English
His interaction with system	Dataflow
His Actions: (Inputs/Outputs, From/to whom)	Structured English
Control constraints on actions	Dataflow
Activity Attributes:	Structured English and as Dataflow annotations
Duration, Frequency	
Operational Cost	
Usage statistics	
* For each Datum produced:	Structured English
Principal structure	
Datum Attributes:	As annotations
Size	
Persistence	

* Major transaction	Dataflow
Typical usage	
Stressful usage	
Transaction Attributes:	As Dataflow annotations
Duration, Frequency	
Usage Statistics	

Such problem domain information:

* Documents the environment the system resides in. Documents system activities.

* Collected from various sources (mission users); they must validate it.

* Aids identification of performance and reliability stress points.

The 1990 RET will aid expressing and formalizing this information. There will be checks for consistency and completeness. Tools will be able to manipulate such information, e.g. for performance prototyping: estimating target system performance on the basis of estimated workloads.

Information on Target System Functionality and Interfaces for Deriving Requirements & Solution architectures

Extrapolating on envisioned 1987/88 VHLL Prototyping tools and RPS functionality, the following can be semiformally expressed:

Information:	Notation for expressing:
1. System Functionality	
* Functionality and a solution architecture that realizes it	* VHLL: Specification by Nested Dataflow and Reusable Modules
2. System Interfaces	
* Graphics, dynamic displays	* RPS: Screen contents have iconic representation also maps, trajectories

Interface & Functionality Descriptions:

* Created by engineer for mission user(s) experimentation, comment, and validation.

* Exercised partially or completely against scenarios developed as indicated below (under "functional model")

Information on Target System Design and Resource Utilization for Analyzing Solution architectures

Currently RPS supports construction of four types of performance models. What makes these RPS models different from VHLL prototypes is that instead of associating executable code with a target system function, a resource utilization profile is associated instead. Against each model, scenarios can be exercised and performance data collected. The models are:

* Model of Analyst - Represented as a set of analyst strategies. A strategy is represented as a set of analyst-dependent procedures. Each procedure is represented as a sequence of screen interactions. Each interaction is represented as a time delay and a target system task; the basis for determining the resources that the interaction consumes.

  Performance can be measured against workstation and processor characteristics; also against analyst characteristics.

* Model of Processor - The processor is represented as a network of computational resources: processors, devices, and data lines. Performance characteristics that are represented include: throughput, capacity rates, transmission characteristics.

  The network is exercised against workload arrival patterns. A workload is represented as a sequence of tasks that contend for specified processor resources. Performance can thus be measured against processor/device/operating system characteristics.

* Model of System Function - The target system is represented as a set of stimulus-thread pairs; in which the stimulus determines the thread. Each thread is represented as a schedule of logical system functions, e.g. database retrievals and input/output operations. To stimulate the model, a scenario can be defined. A scenario is represented as a sequence of events. An event is represented as a set of stimuli.

  The system is exercised against the scenario. Performance can be measured against target system functions characteristics.

* Communications Model - A communications system is represented as a network configuration supporting message traffic. Performance characteristics that are represented include node capacities and flow rates.

  The model is exercised against a message stream. Messages are represented as a source-destination pair. Performance can thus be measured against communications network characteristics.

C.4 General Observations

Updating Goals

Though wishes will change with time, the model indicates there will be no attempt to update the original goals. This is because the goals become folded into the initial requirements, and they get updated there during successive cycles of requirements evaluation. The preferred way to effect permanent updates is thru changes to the domain models referenced.

Flexibility and Generality of the Model

The model is not intended to favor a particular tool or technique within generic categories of support, e.g. prototyping. It is therefore largely independent of any specific tool or technique.

Nor is the model intended to favor a particular order to the requirements engineering activities. Thus, for example, an interface can be prototyped before performance criteria are even established in the requirements.

With a very few exceptions (namely, "Testbed effectiveness" information, "Exit Testbed" and "Measurement of Process and Product" activities), the process model is quite generic and is applicable to requirements engineering in general.

Ambiguity in Definitions

The concepts of goal, requirement, and solution architecture lie in a continuum and defy precise separation; particularly in practice, where such descriptions intertwine. They are more easily characterized thru relative comparisons.

Bounding of the Solution Space

There is a phenomenon called "unexpected bounding of the solution space", in which the mechanisms used to express requirements unexpectedly eliminate some solutions from consideration. The phenomenon is considered in greater depth in Appendix F. Requirements engineers need to be aware that the form of expression they employ, their choice of methodology, and their previous experience may bound the solution space in undesirable ways.

In the long term, requirements engineers should be given tools to help detect implicit assumptions that bound the solution space. Such tools would be categorized under the static analysis activity in section C.2.

APPENDIX D: REQUIREMENTS ENGINEERING TESTBED (RET) TOOLS AND ARCHITECTURE

D.1 Tools Currently Under Development

These are the tools currently under development by RADC:

"Analyst" developed by Imperial College and subcontractor Systems Designers,

"RPS" (Rapid Prototyping System) developed by Martin Marietta Denver Aerospace, and

"VHLL Prototyping Tools" (VHLL = Very High Level Language), developed by International Software Systems.

Analyst

Goal: guide the requirements analyst in the collection, structuring, and validation of information about target system roles and context. Result: a dataflow specification of system requirements organized as a functional hierarchy of viewpoints which represent the processing requirements of the target system, its users, and other systems with which it interacts; performance and reliability are partly addressed.

Approach: the Analyst supports the CORE method by enforcing CORE rules that guide and track the requirements analyst in the use of CORE diagrammatic notations in the statement of system needs and context.

The CORE method consists of rules and procedures that guide the requirements analyst in: factorizing system needs and context into "viewpoints", documenting semiformally each viewpoint, and then identifing performance and reliability needs through consideration of transactions or scenarios that may cross several viewpoints.

Each CORE viewpoint is documented in terms of interactions (actions, data produced/consumed) with the target system and other relevant agents. A viewpoint is also produced for the target system, which documents the functional requirements; as opposed to the other viewpoints, which model expected system usage and the environment of the target system.

Features:

* word processing support is a key component,
* extendible rule base, heuristics,
* incremental consistency and completeness checking,
* Macintosh host and Macintosh-style graphical interface,
* descriptions in terms of directed graphs with textual labels and annotations.

Future research to be pursued (long-range enhancements):

* Construction (through graphical editing): Much leverage through reuse. Reuse involves finding templates, and incorporating (instantiating and/or editing) them.

* Analysis (unsolicited guidance): Need strategic help and active guidance (CORE rules provide mainly tactical guidance). Need better sensitivity to development history and style.

* Navigation & Animation: Support arbitrary browsing (i.e. transitory structuring of the navigational route). Support walk-throughs and direct execution of operational specifications. Support diagrammatic interactions with animation. Support rapid prototyping.

Database implementation: exploits Prolog mechanisms for tuple definition and pattern-matching retrieval. Performance problems are main driver for future enhancements.

RPS

Goal: Provide the Air Force mission user and acquisition engineer with a system requirements validation capability based on rapid configuration of user interface and system performance prototypes. The prototypes are constructed through a minimum development of new software and a maximum reuse of off-the-shelf graphics and simulation modeling packages. The RPS provides access to the modeling approach natural and appropriate to the specific user, and orchestrates the execution of the various models that are produced.

Approach: The following models can be produced:

* Mission analyst workstation model: helps assess performance impacts that alternative workstation technologies and procedures have on the analyst mission.

* Processor model: helps conduct in-depth resource utilization studies on a contemplated architecture under varied work loads. A basis for assessing system performance under various scenarios and identifying resource-critical conditions.

* Function model: helps estimate system performance given an allocation of system functions to system hardware and software resources.

* Communications model: helps assess performance of contemplated network under simulated message traffic.

Features:

* RPS interface and master control hosted on Apollo

workstations,
* Color and dynamic graphics (animation).
* The following things can be modeled and maintained by RPS:
  target system dataflow
  target system database
  target system knowledge base
  display interfaces
  processor and communication architecture
  scenarios that stress the entire target system, just the communication network, or just the mission of one analyst

Future research to be pursued (long-range enhancements):
* Addition of new models.

Database Implementation: The RPS databases reside on different systems: Apollo workstations, the Vax with VMS, and a Prolog/Lisp processor. No attempt will be made to integrate the various databases. The RPS user can access these databases:

* D3M database (Apollo) - Used to model target system functionality and control. Features: menu-driven schema definition, report formatter, and a forms generation package supporting the interactive creation of records.

* KEE database - builds/maintains knowledge bases in support of requirements analysis of decision support systems.

* Switch table database - maintains code fragments which associate functionality (e.g. database retrieval, simulation model execution, dynamic graphics) with a selected region (e.g. graphic depiction of a system component, portion of a user interface prototype) of the display.

* Support tools database - word processor, spreadsheet, report generator, and version manager for actual production of requirements documentation.

VHLL Prototyping Tools

Goal: Provide a capability to rapidly and minimally describe a program that executes certain target system function(s).

Approach: a VHLL (Very High Level Language) is used for specifying prototypes. The VHLL prototyping environment provides support for: reuse of application-specific modules, rapid modification, and collection of performance statistics during execution. Both object-oriented and stimulus-response specification approaches can be used.

Features:
* Hosted on Apollo workstation.
* A browser interface to prototyping and project data.

Instantiating mechanisms facilitate reuse.
* Option for graphical entry of VHLL specifications.

Future research to be pursued (long-range enhancements):
* VHLL: enhance with a richer standard set of data and function operators.
* Prototyping environment: embed within a general design database in which programs would be directly synthesized. Tools, display would be initiated by database activity.

Database approach:
* Database: will maintain function interface/body descriptions, data types, prior designs, classifications, queries.

* Data model: an object-centered data model supporting CAD needs for dynamic metadata and data with complex attribute values. Query language: selection allows navigation over relationships. Support for updates thru views.

* Design management: maintain consistency of designs and tracks dependencies. Versioning: will be fine-grain; there may be multiple active versions. Configuration management to manage consistent assemblies of objects.

* Strategy: Utilize commercial CAD database. Database is system focus for all synthesis, analysis activities.

D.2 Loose Coupling Strategy

A very loose coupling between the contractors' tools is feasible within the time and scope of the current tool development efforts.

Loose coupling means that the data produced by one tool can be input to another with only minimal (perhaps no) effort and minimal (perhaps no) loss of information. Loose coupling will allow the testbed user to take (say) a functional dataflow representation created by one tool and provide it as input to another tool.

Loose Coupling game plan:

(1) Each contractor will provide an object analysis on his tools, identifying objects, attributes, relationships, operations, and aggregation operations (building bigger objects).

(2) Identify semantic gaps in the different object analyses. (A semantic gap between two tools indicates precisely what kind of information they cannot share and which must be lost in moving shared data between them because of a lack by one of the tools of the necessary semantics for representing the information.) Where few gaps exist, produce an object schema overlapping all three tools.

(3) The common schema will be the basis for determining the kind of file conversions that must be provided, in order to move information successfully from one tool to the other.

(4) Other support:
* for file transfers between Macintosh (hosting the Analyst) and the Apollo (hosting RPS, VHLL tools).

* some configuration/version management assistance across the tools is desired.

D.3 Integration Strategy

To significantly reduce the amount of effort required to achieve integration, the panel defined a strategy of things to be done in the very near term by RADC and its tool contractors. The strategy is to: (1) Define standards for the final integrated RET of 1990. (2) Make current tool contractors aware of these standards to influence the design and implementation of their tools. (3) Encourage and maintain informal communication between the tool contractors and RADC to help track problems associated with the use of these standards.

To achieve (1) and (2) above, all current tool contractors were invited to a meeting of the panel to help define the standards. Conclusions are presented below. To help achieve (3), contractor meetings to discuss database issues have been recommended to begin in early 1987.

Following the discussion on standards, the loose coupling plan (of section D.2) is contrasted with the integration strategy.

User Interface Standards and Strategy

Assumption: If all current tool contractors can agree to a similar interface style now (i.e. how to invoke tools, how to manipulate data), the later integration will be much smoother, and in the interim, users going from one tool to another will not have to fundamentally change the style in which they access requirements and tool functionality.

Current tool interfaces: Graphical editing plays a key role in every contractor's approach to the tool interface.

Standards to be followed:

(1) Macintosh (Apple™) Style Manual documents syntactic conventions for menu systems and supports the paradigm of direct manipulation at the interface. Conventions outlined in this manual are to be used by tool contractors to achieve a common syntax to their menu systems.

Standards desired:

(1) Paradigm of resource files (Windows™): conventions for exploiting menus, dialogue boxes, icons, and accelerators for the purpose of creating programs with adaptable interfaces (i.e. without need of recompiling).

(2) On commands: consistency of application-specific windows (e.g. the command bar) and the commands (and their options) invokable within them (e.g. same file operations and operation names).

(3) On "gestures": consistency in the implicit ways of getting things done (e.g. expand/contract boxes).

(4) Standardization vehicle: Systems Designers can forward appropriate Analyst code/conventions to the other tool contractors as a guide.

The panel's user interface integration strategy is to have a common editor/browser interface to the objects, through which different actions can be selectively invoked. This is consistent with the panel's recommended RET architecture.

Database Standards and Strategy

Assumption: The different contractors are developing tools that need to access (share) the same data.

Thus standards are needed for:

(1) Data Model.

* How is the data structured? What part is formal structure (e.g. arcs), and what part is text (e.g. nodes).

* How is the data accessed? Need standards on the query and programmatic interfaces.

* What atomic data and database operations on these data are assumed? Example: establish/delete node/arc.

* Associative retrieval: need standards on how to navigate associations and whether it is permitted in both directions.

(2) Schema.

Should the schema itself be represented in the database so that tools can operate on it (e.g. to track inheritance)? What are the performance implications of this strategy (i.e. each data access requires a schema access)?

(3) Rules and Inferencing.

Types of rules: (1) design constraint rules to maintain the consistency of the objects being built, (2) transformation rules which permit derivation of implied information from multiple relationships in the data, and (3) rules as a form of program encoding (capturing programs in an incremental and modifiable way).

* Rule representation: How should they be declared? Should they be embodied in the code or represented in the database to be manipulated as data?

* Support which: derive-as-needed or truth maintenance? (Given rule A,B --> C, and A,B, do you assert C and store it in the database or derive it as needed?)

* How to turn rules on/off?

(4) Triggering.

When a database update matches a particular pattern, do some processing. Example: update screen as result of a change.

* Similar issues as in "Inferencing" above.

* Two different notions of "transaction":

  ** atomic transaction: "batching" all updates to avoid triggering rules until completion, and

  ** interactive transaction: updates are immediate, but if an abort occurs before completion of the transaction, the database is restored to the pre-transaction state.

(5) Import/export facility.

An import facility allows inputing data produced by another tool in some foreign format. An export facility allows taking out data in a format (e.g. ASCII) that can be used by other tools.

Problems and future research issues:

* Limited database size: Artificial Intelligence databases are capable of handling small numbers of facts (about 30K). How does their performance scale up?

* Performance:

  ** Strategies: Cache. Optimize for retrieval operations. Limit size of the problems to be

addressed by the tools (i.e. in the context of RET experiments).

** Performance problem: The problem of determining the context in which to bring something in from disk is exacerbated by the large number of rules. Inheritance mechanisms and having the schema represented in the database further exacerbates this problem.

Loose Coupling (section D.2) vs. Integration

Availability to testbed:

Loose Coupling: Early 1988, with nominal effort.
Integration: 1990, involving considerable effort.

Degree of common data representation:

Loose Coupling: Perhaps none, file conversion probable.
Integration: Common representation for all data shared. Storage/retrieval thru a common database.

Shifting between tools:

Loose Coupling: Limited in frequency, as conversion results in some loss of semantic content and affects performance.
Integration: Frequent. The RET user will be able to take full advantage of equal and uniform access to all tools and objects, yielding, for example, improved prototypes and increased productivity.

D.4 Long-Range Architecture

We present below a long-range (at least ten years) software architecture for the RET. The architecture is made up of three parts: a user interface (the "RET editor"), a database (the "RET database"), and a language (the "RET language").

The RET architecture consists of:

RET editor - the user interface to all documents and tools. The interface can be: customized (e.g. domain-specific, dialog style, graphics vs. text), syntax-directed, and tolerant to ambiguity. Capabilities for:

(1) Browsing (finding) relevant objects.

(2) Viewing (selection/presentation) objects.

(3) Annotating objects with comments, questions, etc.

(4) Invoking all RET tools.

(5) Multiple windows/contexts/user access.

(6) Direct editing of relevant objects, WYSIWYG style.

These capabilities require close coupling between the RET editor and RET database.

RET Database - the common repository for all requirements-relevant information, all RE tool artifacts, including evolution. Features:

(1) Object definition facility, based on the RET language.

(2) Query/update interface, supporting associative retrieval.

(3) Rule-based architecture. Provision for defining rules to obtain consistency, effect-propagation, and automation. Is fundamental to long-range RET goal: a WYSIWYG suppressed-tool interface, in which tools that process objects are directly invoked by state changes, as indicated by the rules.

(4) View management. Support for view customization.

(5) Performance scales with size well; simultaneous access.

(6) Multi-language: available to all RET tools.

RET Language - a single formal wide-spectrum language for expressing requirements, specifications, scenarios, and domain definitions. Goals and partial descriptions are given formal semantics. Support required:

(1) customization of the user interface to make the language usable to a broad user class (RET editor),

(2) support and tracking of incremental modifications to descriptions (RET database and RET editor).

APPENDIX E: DETAILED DESCRIPTIONS OF RET R&D ISSUES

This appendix contains detailed characterizations of all R&D issues that form part of the RET R&D program.

RET R&D program research issues and development issues are given respectively "RET Research Issue" and "RET Development Issue" characterizations, and have the following format:

which track (formal language vs. evolutionary),
problem description,
value in requirements engineering process,
solution approach,
5-year objectives,
10-year objectives,
risk (likelihood of results),
cost (degree of effort required),
contingencies (dependencies),
conclusion (recommendations), and
author name (which panel members wrote the description).

Those issues considered to be the best candidates for funding were also given "RET R&D effort" characterizations. An RET R&D effort characterization has this format:

objective (what is to be accomplished),
scope (effort boundary, technical areas to be addressed and those which will not),
technical approach (how the objective will be achieved),
background (relevant terminology, related work toward this effort's objective, etc.),
references,
duration (in months),
cost (in dollars)
deliverables (product and date (i.e. months from effort start), and
roadmap (task schedule against 1986-1995 timeline).

Costs include capital costs for equipment and resources. These estimated costs reflect the panel perception that these R&D efforts require the highest quality people and laboratories.

Section E.1 characterizes the R&D issues addressed in the evolutionary track. Section E.2 characterizes the R&D issues addressed in the formal language track. As all formal language track issues contribute to the same goal, namely the formal treatment of requirements, a "research issue" characterization and roadmap is given for the whole track; individual issues are only given R&D effort characterizations.

E.1 Evolutionary Track

E.1.1 Goals and Requirements Synthesis

RET Research Issue. Evolutionary Track.

PROBLEM DESCRIPTION:

The synthesis activity takes unstructured textual expressions of need as inputs (often conflicting with each other) from different sources and produces as output a formal structuring of the input texts (which have been refined, made feasible, and conflicts resolved). Synthesis is a complex activity, often a group activity, and for the foreseeable future, the challenge is one of supporting manual synthesis.

VALUE IN REQUIREMENTS PROCESS:

Explicit requirements statements have a critical communication role between and among system definers, designers, and reviewers. They form the end product of the RET activities.

SOLUTION APPROACH:

(1) Reuse mechanisms: The RET analyst will need to access, modify, and incorporate existing requirements descriptions. This requires mechanisms for browsing previously-synthesized descriptions, and composing them into the new synthesis context.

(2) Viewing mechanisms: As synthesis is largely manual, the RET analyst will need to see all relevant information (mission viewpoints, domain information, and related descriptions). Some (possibly most) information will be developed using domain-specific terminology and notations. Controlling information changes and presentations thru different perspectives of a group is largely a problem of view management.

(3) Syntax-directed editing mechanisms: Syntax aids and checking should be well integrated with editing, supporting the synthesis of the formal parts of goals/requirements descriptions.

5 YEAR OBJECTIVES:

Provide an RET database of appropriate data definition, query/update interface, and view management facilities.

Enhance the Analyst: Research, develop, and integrate better syntax-checking, view management, and reuse mechanisms; base on top of the RET database.

10 YEAR OBJECTIVES:

Improved classification of requirements, domain information, etc., improving their ease of accessibility.

Develop multi-language synthesis capability: enabling a heterogeneous group of people to develop portions of the same set of requirements utilizing different domain languages.

RISK:

To the extent the objectives can be achieved thru database technology, there is a high probability of success. Otherwise, there is significant risk.

COST:

1-2 man years for 5-year objectives. 2-3 man years for the 10-year objectives.

CONTINGENCIES:

Finding an appropriate RET database in less than 5 years.

CONCLUSION:

To the extent that the viewing mechanism can be provided by a DBMS, it should be brought into the RET with no specific research funding. Where new research is needed, it is not clear that the technical opportunities are well understood, so funding cannot be recommended at this time.

AUTHOR NAME:

Michael Konrad, Terry Welch.

E.1.2 Domain Models and Information

RET Research Issue. Evolutionary Track.

PROBLEM DESCRIPTION:

Domain information can be: (1) persistent - useful to more than one project and/or for long periods of time, and (2) a basis for decisions - key assumptions. But domain knowledge is often in people's heads (e.g. mission specialists). Clearly, collecting and organizing domain information and then making that information accessible to both RET engineers and tools presents enormous challenges and opportunities.

VALUE IN REQUIREMENTS PROCESS:

The most significant role played by domain information is as assumptions or facts about the environment in which the target system is to reside (e.g. user characteristics and expectations).

SOLUTION APPROACH:

The technical approach must focus on the use of domain knowledge in requirements engineering; much effort is already being spent on the general problem. The effort must determine: (1) what domain knowledge is used, (2) how it is used (for what purpose (including by tools) and with what frequency), and (3) how it changes (nature and frequency of change). In other words, build a usage model of domain knowledge in requirements engineering. Only then can a practical structure and representation for domain information be selected.

Ways of getting new knowledge into the model (knowledge acquisition and classification, domain-specific language/interfaces) and getting it out (interfaces with tools, dor expert critiquing, and domain-specific language/interfaces) wi e tailored to these results but need investigation. It is not yet c ear which input/output approach will provide the best leverage. An input/output approach will be recommended at the conclusion of the study described in the preceding paragraph.

5 YEAR OBJECTIVES:

Selection of domain model structure:
* structure driven by key requirements engineering concerns
* goals/requirements/solution architecture descriptions and

their rationalizations can reference
* basis for recommending input/output approach to getting domain information in/out

Build models of one or two key domains

Adapt tool interfaces to permit access

10 YEAR OBJECTIVES:

Enhance RET tools to better interpret and use domain knowledge.

Build a domain information input/output mechanism(s) out of some combination of:
* classification scheme for automatic acquisition,
* domain-specific language/interface,
* domain experts that can assist the engineer in:
eliciting requirements relevant to that domain and
in the use of domain knowledge

Augment/revise domain model structure to capture key domain information required by new tools.

RISK:

Low to moderate risk. Understanding the general role of domain knowledge in requirements engineering, both present and future, is key to success. Failure to understand that role may lead to capture of irrelevant information. New tools may out date the usefulness of the knowledge represented. Domain knowledge is also subject to change with improved theories. Automatic acquisition is largely a classification problem, and poses some significant risk.

COST:

Degree of effort required: 5-year objectives: 4 man years for first two objectives, 3 man years for last one. 10-year objectives: 10 man years.

CONTINGENCIES:

Tied to evolution of tools and domain theories.

CONCLUSION:

The domain model approach offers much promise, even if automatic domain knowledge acquisition should prove very hard and therefore remain significantly manual. Recommend funding.

AUTHOR NAME:

Michael Konrad, Terry Welch.

RET R&D Effort

ISSUE NAME: Domain Models/Information

OBJECTIVE:

Define a domain modeling technology:
(1) helps organize domain information into a model,
(2) for access by tools, and references by Goals and ... Requirements descriptions,
(3) for communicating domain information to/from humans,
(4) supports domain model evolution.

Interface tools to domain models. Enhance tool use of domain information.

SCOPE:

Technical areas to be addressed: Domain models, Knowledge acquisition. However the technical approach is not to advance generic technologies in these areas but to develop new technologies addressing the most pressing problems in the use of domain information in requirements engineering.

TECHNICAL APPROACH:

Determine the usage of domain information in requirements engineering as a basis to identifying where the best leverage lies (what to capture and how to structure if for later access). Build domain-specific languages and/or interfaces for the most common domain(s) to aid in capturing the domain information relevant to an application. Address relevant database issues: synergies between different domains, aggregation and classification, versioning, and evolution. Enhance appropriate tool interfaces. Investigate role for expert systems that use domain information to help elicit and validate requirements.

BACKGROUND:

The "Analyst" [1], a requirements engineering tool under development by Systems Designers (an effort sponsored by RADC), already provides a limited formal modeling capability. Imperial College, under contract to RADC, is exploring enhancements that would increase the conceptual modeling capability of the tool [2].

Several reports discuss the capture, use, and/or evolution of domain knowledge in software-related activities: requirements engineering [3], automatic programming [4]. Several reports discuss the development of tools to aid encoding domain knowledge in the creation of "friendly" domain-specific interfaces and

languages: for domain-specific requirements languages [5], and for the construction of software systems [6] and [7]. Reports that stress the role and value of domain modeling as a precursor to the "requirements specification" include [3] and [8].

REFERENCES:

[1] M. Stephens and K. Whitehead, "The 'Analyst' - An Expert Systems Approach to Requirements Analysis", draft.

[2] A. Finkelstein and C. Potts, "Structured Common Sense: The Elicitation and Formalization of System Requirements", draft.

[3] S. J. Greenspan, J. Mylopoulos, and A. Borgida, "Capturing More World Knowledge in the Requirements Specification", Proc. 6th Int. Conf. Software Eng. Also other papers by the authors.

[4] David R. Barstow, "Domain-Specific Automatic Programming", IEEE TSE, November 1985.

[5] Alan M. Davis, "The Design of a Family of Application-Oriented Requirements Languages", IEEE Computer, May 1982.

[6] James M. Neighbors, various Draco papers.

[7] S. Sundfor, two reports on application of Draco: "Draco Domain Analysis for a Real Time Application ..." Irvine TRs: RTP 015-016.

[8] Bartlett, Cherrie, Lehman, MacLean, and Potts, "The Role of Executable Metric Models in the Programming Process", Imperial College of Science & Technology.

DURATION:

Domain modeling Study	30 months
Select domain model & build example(s)	
Interfacing tools to domain model	18 months
Enhanced tools	(part of separate tool efforts)
	18 months
Build input/output mechanism for domain information	
	30 months
Evolve domain model structure	24 months

COST:

Domain modeling Study	.6 million
Interfacing tools to domain models	.45 million
(attached to "Tools integration contract")	
Enhanced tools	.45 million
(attached to "Tools integration contract")	

Build input/output mechanism:	.75 million
Evolve domain model structure	.3 million

DELIVERABLES:

PRODUCT	DATE (months after start):
Domain modeling Study	
Domain Model Structure Design	18 months
Build example domain models	27 months
Domain Model Structure Final Design	30 months
(Attached to integration contract)	
Design/deliver interface to domain models	18 months
Enhance to better exploit domains	36 months
Build input/output mechanism	
Design	18 months
Delivered code	30 months
Evolve domain model structure	
Revised Domain Model Report	16 months
Design and delivered code modifications	24 months

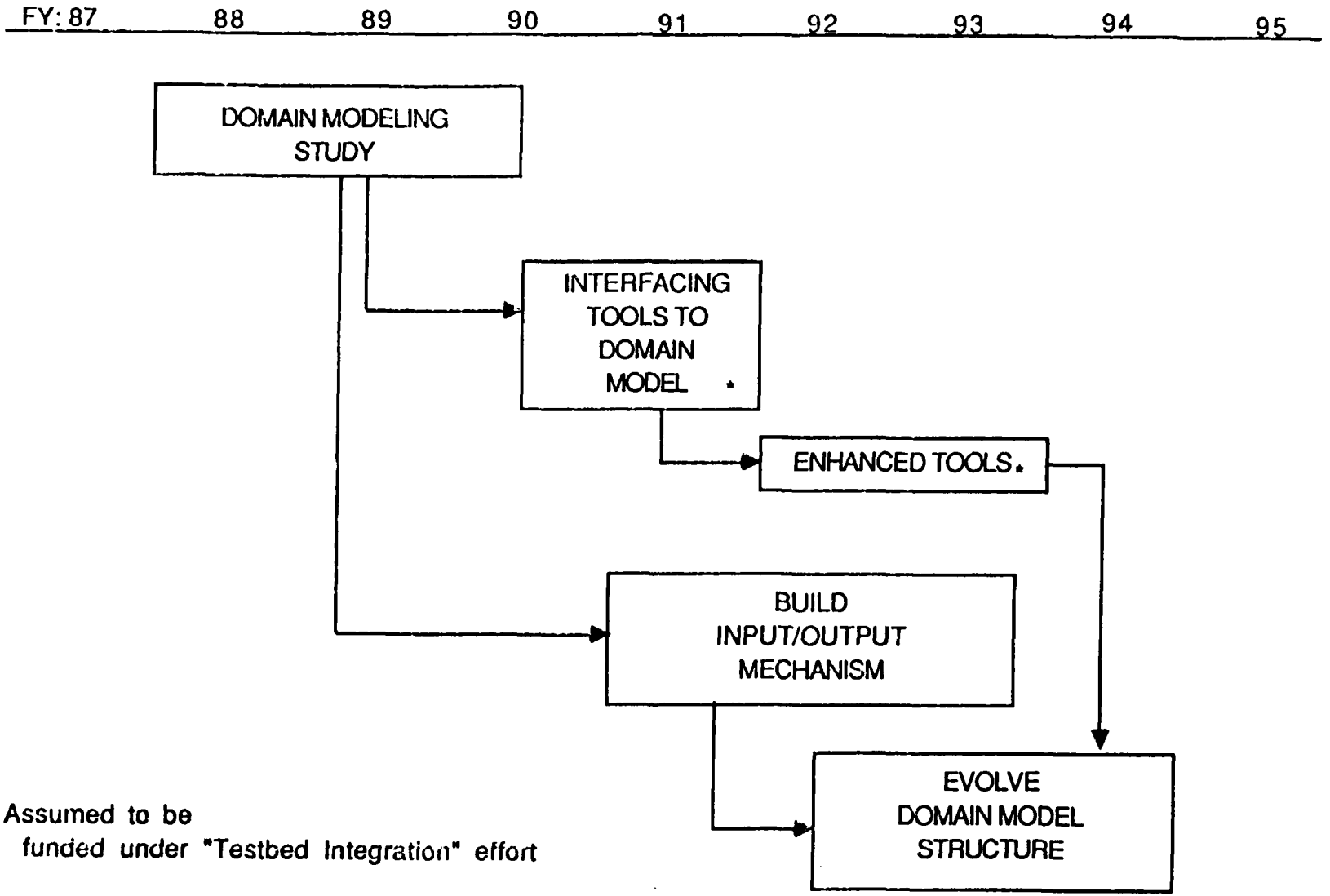


Figure E.1.2-1 DOMAIN MODELS R&D EFFORT

E.1.3 Requirements - Static Analysis

RET Research Issue. Evolutionary Track.

PROBLEM DESCRIPTION:

Analyze the requirements without executing a prototype of the solution. Provide checks for completeness and consistency. Animate the requirements.

VALUE IN REQUIREMENTS PROCESS:

Requirements analysis at an early phase avoids difficulties associating requirements with solutions and reduces the need for scenarios. Static analysis removes inconsistencies at an early stage in requirements engineering.

SOLUTION APPROACH:

Convert the requirements to a controlled natural language. (A near English-like language with a restricted vocabulary). Check the expressions in this language for consistency and completeness. Through a domain-specific meta-knowledge base question untestable, contradicting or redundant requirements. Animate the requirements by allowing editing based on graphical representations of the controlled natural language expression organized along differing categories.

5 YEAR OBJECTIVES:

Develop a controlled natural language and accompanying meta-knowledge base.

10 YEAR OBJECTIVES:

Continue 5 year development with the addition of user interfaces and analysis tools.

RISK:

Excellent chance of results.

COST:

6 man years for the 5 year objective.
3 man years for the 10 year objective.

CONTINGENCIES:

Dependent on controlled natural language.

CONCLUSION:

Fund research at a low level. Not top priority.

AUTHOR NAME:

Stephen Sherman

RET R&D Effort

ISSUE NAME: Requirements - Static Analysis

OBJECTIVE:

Provide tools to analyze requirements without executing a prototype of the solution.

SCOPE:

The effort is bounded by the need to formalize the requirements language. As more semantic and syntactic restrictions are placed on the requirements, we approach the concepts of the formal track and may borrow from the formal track effort.

TECHNICAL APPROACH:

Provide the minimum formalization necessary to carry out the solution approach. If the funding for the formal track is not timely, this effort may provide a start on the formal tack.

BACKGROUND:

The relevant technology is the theory of languages and logic (consistency and completeness).

REFERENCES:

For animation: CORE/Analyst development system by Imperial College and Systems Designers. For Controlled Natural Language: "Requirements for Mechanical Translation - Problems, Solutions, Prospects" in Feasibility Study on Fully Automatic High Quality Translation, (Stachowitz, Bar-Hillel, Winograd, Bob Simmons) Austin, Texas, Linguistics Research Center, The University of Texas at Austin, RADC-TR-71-295, 1971.

DURATION:

Controlled Natural Language (CNL) - 3 years
User Interfaces and Analysis (UIA)- 2 years

COST:

Controlled Natural Language (CNL) .9 million
User Interfaces & Analysis (UIA) .45 million

DELIVERABLE:

CNL	Design	6 months
CNL	Prototype	18 months
CNL	System	36 months

UIA	Design	6 months
UIA	System	2 years

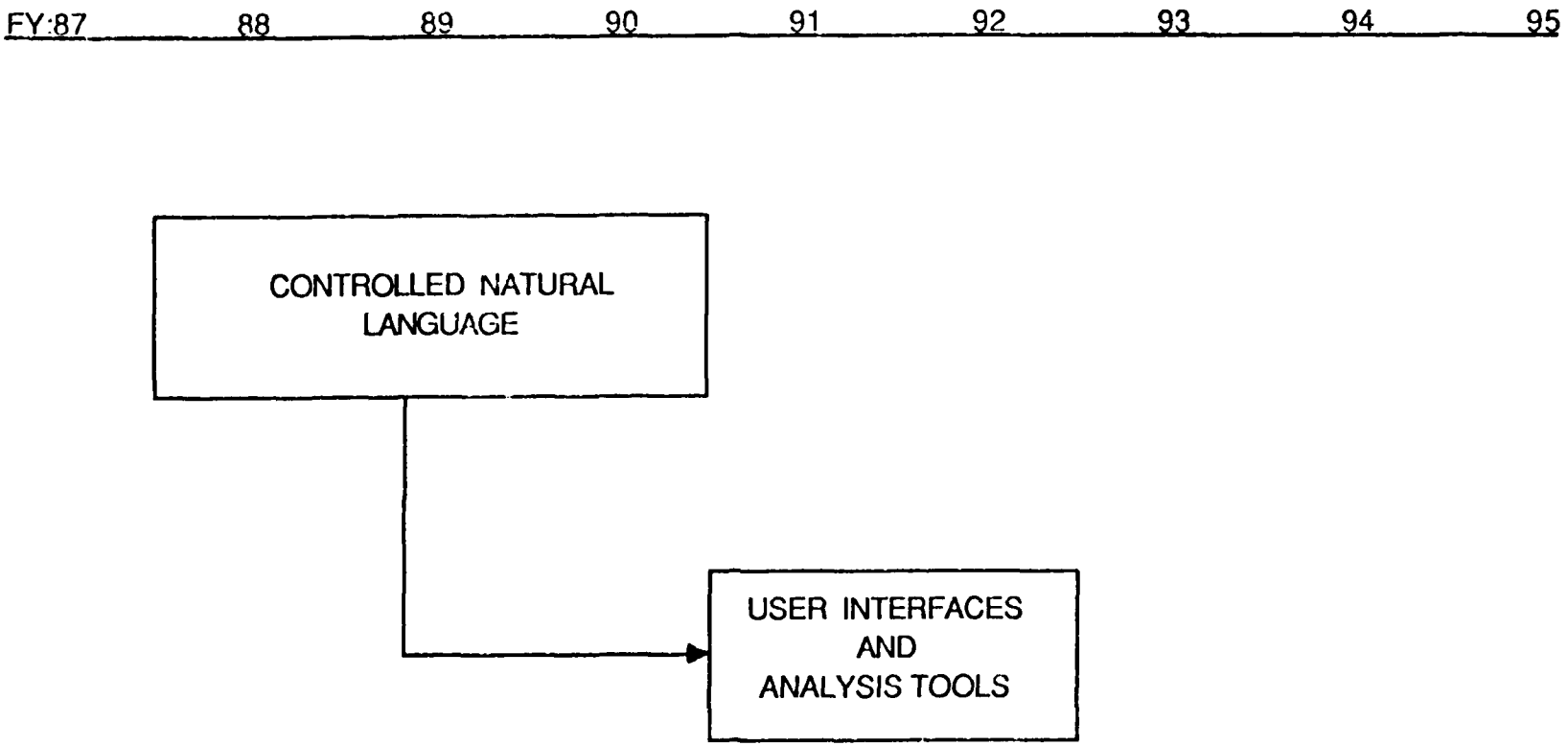


Figure E.1.3-1 REQUIREMENTS - STATIC ANALYSIS R&D EFFORT

E.1.4 Requirements Analysis Methodology

RET Research Issue. Evolutionary Track.

PROBLEM DESCRIPTION:

As a result of both RET experience and outside advances, new requirements analysis capabilities will be brought into the RET. The RET requirements engineer will need guidance, e.g. methodologies, in the intelligent application of these new RET capabilities to solve his requirements problems.

VALUE IN REQUIREMENTS PROCESS:

Intelligent guidance in the application of new and better tools and techniques will lead to automation of more requirements analysis resulting in improved requirements and productivity.

SOLUTION APPROACH:

The problem of giving guidance to requirements analysis activities will be attacked two ways.

In the near term, the CORE method will be extended to include guidance in the use of currently-planned tools (e.g. the "RPS" and "VHLL Prototyping tools").

In the long term, the role of CORE in guiding requirements analysis activities needs to be reexamined; other methodologies should be considered. The selected methodology must guide all engineering activities that help effect analyses on requirements, e.g. solution architecture synthesis to derive cost, risk, and performance estimates.

5 YEAR OBJECTIVES:

Extend the CORE method:
* into solution architecture synthesis and analysis, and
* to provide guidance in use of RPS and VHLL Prototyping tools.

10 YEAR OBJECTIVES:

Provide an RET requirements analysis methodology:
* structures analysis activities to meet RET user objectives,

* guides solution architecture synthesis, and
* supported by RET formalisms and tools.

RISK:

Moderate risk. Selection of the best methodology will be difficult because of the difficulty in determining the relative effectiveness of tools and techniques.

COST:

Degree of effort required: 5-year objectives: 2 man-years. 10-year objectives: 6 man-years.

CONTINGENCIES:

Knowing the good vs. bad tools and techniques helps one define a good methodology. Thus strongly dependent on success in the "Testbed effectiveness" R&D effort. Testbed tools effectiveness measures must be developed and RET usage data collected both on the tools and on the applications and purposes of use.

CONCLUSION:

Good guidance in the use of RET tools is a key to successful requirements analysis. We recommend funding this effort.

AUTHOR NAME:

Michael Konrad, Terry Welch.

RET R&D Effort

ISSUE NAME: Requirements Analysis Methodology

OBJECTIVE:

Provide RET users guidance on the use of new RET tools, through:

(1) Near term: extending the CORE method into strategies for using RPS and VHLL Prototyping tools.

(2) Long term: select a methodology that directs all engineering activities related to requirements analysis.

SCOPE:

Technical areas to be addressed: Requirements Analysis, Requirements Methodology.

TECHNICAL APPROACH:

Near term: Do (1) below. Then select those CORE steps that can be made more effective through utilizing RPS and AND Prototyping tools capabilities. Place hooks in the Analyst to indicate when to use these prototyping tools and pass data to them.

Long terms: Do (1), (2) and (3).

(1) Analyze each RET tool, identifying what it analyzes and what is produces. Assess the utility of each tool by analyzing RET tool effectiveness data if available.

(2) Identify a few candidate RET methodologies (in particular, CORE); extend them to appropriately utilize the tools determined in (1), identifying tool entrance and termination conditions. Determine experimentally their relative strengths and weaknesses. Select one methodology to be the "RET methodology".

(3) Build a tool that indicates to the RET user which RET tools are appropriate at each stage (if more than one, what value and effort are implied), consistent with the RET methodology.

BACKGROUND:

The "Analyst" [1] is a tool supporting CORE requirements specification and analysis activities. It currently does not address prototyping.

SREM [2] was an early requirements methodology. In a recent evaluation [3], SREM was considered to be best used when defining, correcting, or analyzing software specific requirements; i.e.

after the (traditional) system requirements analysis phase had been completed, and long after the conceptual phase (e.g. feasibility assessment and trade-off analysis). The methodology proposed by this research would address all of these.

REFERENCE:

[1] M. Stephens and K. Whitehead, "The Analyst - A Workstation for Analysis and Design", Proc. 8th Int. Conf. Software Eng., IEEE Comp. Soc. Press, 1985.

[2] M. Alford, "A Requirements Engineering Methodology for Real Time Processing Requirements", IEEE T.S.E. SE-3(1).

[3] A. Stone, D. Hartschuh, and B. Castor, "SREM Evaluation", Rome Air Development Center, RADC-TR-83-314.

DURATION:

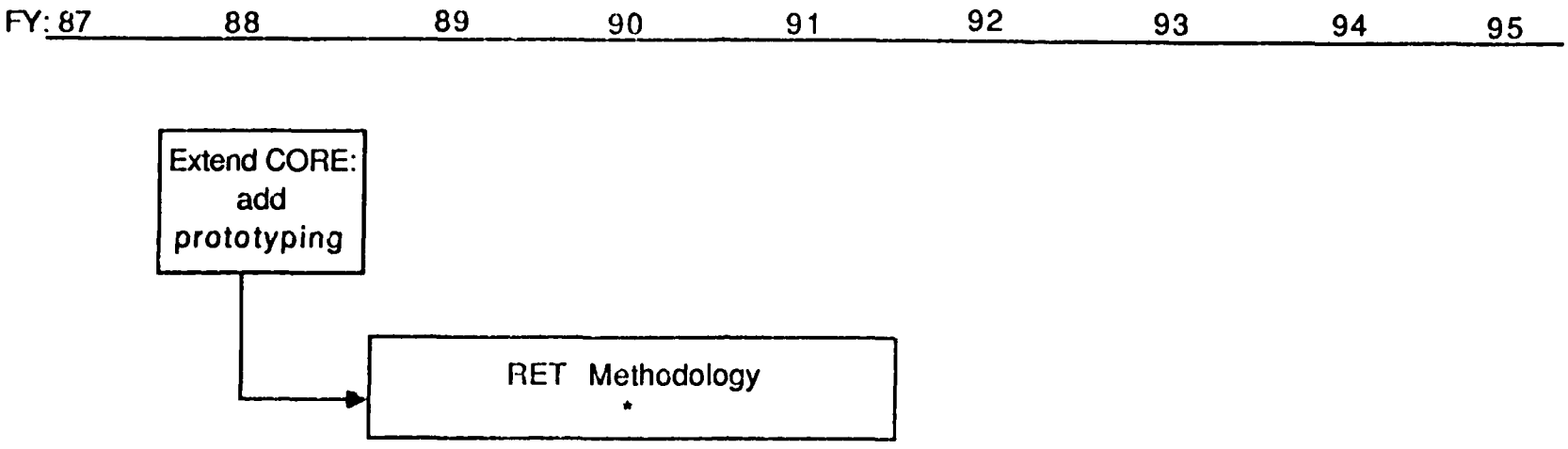
Extended Core: add prototyping	12 months
RET methodology: all analysis activities	36 months

COST:

Extended CORE: add prototyping	.3 million
RET methodology	.9 million

DELIVERABLES:

PRODUCT:	DATE (months after start):
Extended CORE:	
Report on adding prototyping	12 months
Delivered code:	
Hooks in Analyst to prototyping tools	12 months
RET methodology:	
Report on RET tools:	
Scope and Effectiveness	12 months
Report on methodology comparison	24 months
Delivered code:	
Indicates which RET tools to use	36 months



*Requires access to RET facilities

Figure E.1.4-1 REQUIREMENTS ANALYSIS METHODOLOGY R&D EFFORT

E.1.5 Dynamic Analysis

RET Research Issue. Evolutionary Track.

PROBLEM DESCRIPTION:

The dynamic analysis of requirements constitutes the most valuable means for validating a candidate set of requirements. It provides end users and system specifiers with feedback and allows them to "exercise" certain characteristics of a proposed new system. This early feedback will greatly enhance the quality and fidelity of future systems.

VALUE IN REQUIREMENTS PROCESS:

As referenced above, early feedback on the validity and accuracy of system requirements is highly desirable. It will enable substantially more of the logical "What if-ing" that leads to improved systems designs and implementations. Early detection of faulty assumptions and assurance that the requirements are clearly and concisely represented will result in considerable cost and time savings for major programs.

SOLUTION APPROACH:

This dynamic analysis of requirements is clearly very open. A number of capabilities are suggested. A user would clearly like to be able to investigate the cohesiveness and validity of a candidate set of system requirements. The use and continued development of rapid prototyping systems to study a system is clearly a part of what is required. Augmentation of initial prototyping capabilities with the ability to "exercise" a proposed system and study the interrelationships of its various requirements is highly desirable. Packaging this type of animated prototyping capability in a manner in which end users and their analysts can study a proposed system is the direction of most promising pursuit.

Augmenting these prototyping capabilities with domain-specific knowledge based concepts is a more long-range undertaking. Developing techniques for representing and analyzing new system requirements versus a known domain model and browsing through various interrelationships could be very valuable. A phased approach to this is suggested since there are clearly quite a few aspects to this overall problem. The first phase of this activity will be aimed at determining the best ways to capture and represent the user's problem domain with suggesting approaches to

its analysis. Following a go/no-go decision, much more effort will be focused on building a more powerful browsing and analysis capability. This advanced support system would allow the building of much more complete user domain models and provide more powerful "logical browsing" and analysis capabilities for more completely validating end user requirements.

A central goal in all of these activities is to provide a much more flexible environment for specifying and analyzing higher level requirements and specifications. By providing more rapid and valid feedback at an early stage in a project vastly superior systems can be developed with substantially less risk.

5 YEAR OBJECTIVES:

Develop general purpose prototyping capabilities that will allow a user at a very high level to graphically specify a series of tasks and/or activities and some of their important interrelationships and "execute" thru animation the flow of information and control between tasks.

10 YEAR OBJECTIVES:

Develop knowledge-based systems which allow end users and their support specialists to specify attributes of their problem domain and query a system which "exercises" queries about related attributes and specifications for new systems. For example, dynamic browsing is of the form, "I am a pilot charged with carrying out a certain type of mission, what requirements might be affected by this particular type of mission?"

RISK:

Rapid prototyping systems are becoming available already, thus the risk associated with the 5 year activities seems quite minimal.

The longer range activities are much more questionable since they really depend rather heavily upon significant advances in knowledge collection and manipulation.

COST:

Rapid prototyping and animation activities should be sponsored with a 5 - 10 person year effort in the first 5 year time frame.

An intelligent knowledge-based capability should be investigated in phases to determine its feasibility and expected payoff. An initial effort of 2 person years should be supported to be followed up with a go/no-go decision for a subsequent 5-10 person year effort in the 10 year time frame. (If something reasonable can not come from this level of effort - we are probably trying to pioneer the field too much.)

CONTINGENCIES:

The prototyping activities are currently tied to various representations and methodologies. Since it is not clear there exists any UNIQUE desirable methodology we should attempt to develop tools that are adaptable to methodological changes. It is also clear that graphical methods will continue to evolve and we must be supportive of such changes as well.

As we look into more sophisticated knowledge-based approaches to representing and analyzing high level specifications, we become very dependent upon advances in the representation, collection, and manipulation of these knowledge bases. We should focus on the use of this technology rather than sponsoring its research directly.

CONCLUSION:

Support for enhanced rapid prototyping is clearly doable and valuable. Exploration of techniques for providing more user "animation" types of capabilities in conjunction with rapid prototyping is also highly desirable.

A phased approach should be applied to the longer range knowledge-based type activities since their feasibility and payoff is not yet clear.

AUTHOR NAME:

Leon G. Stucki

RET R&D Effort

ISSUE NAME: Dynamic Analysis

OBJECTIVE:

The objective of this research area is to develop improved methodologies and tools for analyzing and validating system requirements at a very early life cycle phase. Clearly the more feedback, iteration, and analysis that can be performed within the requirements activity itself, the more cost and schedule savings can be realized.

SCOPE: Two types of activities are included in this area.

The first involves the development of enhanced prototyping capabilities. Most of the ideas involved here have in fact been demonstrated to varying degrees already. No real surprises are anticipated in their further development and application on this effort.

The second focus area for research is less well defined. Although the desire to apply knowledge-based technology to this problem appears useful, the practicalities of this technology have yet to be totally proven.

TECHNICAL APPROACH:

The proposed technical approach will be evolutionary in nature. Initial emphasis will be placed on incorporating "known" techniques and types of tools within a framework suitable for requirements analysis activities. This framework will be augmented with increasingly powerful analysis capabilities. Hopefully, with time, "smart aids" will serve as guides to the user in better representing and analyzing future system requirements.

The application of dynamic analysis to requirements engineering, as a general area of investigation, is clearly very open ended. A number of capabilities are suggested. A user would clearly like to be able to investigate the cohesiveness (consistency & completeness - validity) of a candidate set of system requirements.

Task 1:

The use and continued development of rapid prototyping systems to study a system is clearly the first task to be included. Augmentation of initial prototyping capabilities with the ability to "exercise" a proposed system and study the interrelationships of its various requirements is highly desirable. An integrated prototyping system will be designed and built incorporating this

animation capability and packaging it for use by end users and their analysts.

Task 2 [Phase I]:

Augmenting these prototyping capabilities with domain-specific knowledge-based concepts is a more long-range undertaking. Developing techniques for representing and analyzing new system requirements for a known domain model and browsing through various interrelationships can be very valuable. A phased approach to this is suggested since there are clearly quite a few aspects to this overall problem.

Task 2 [Phase II]

Thus, the second major phase of this activity will be aimed at determining the best ways to capture and represent the user's problem domain with suggested approaches for its analysis. A moderate feasibility study and initial demonstration task will be followed up with a go/no-go decision. Assuming a favorable decision, a subsequent effort will focus on building a more powerful browsing and analysis capability. This advanced support system will support the building of much more complete user domain models. It will also provide more powerful analysis and "logical browsing" capabilities for validating end user requirements.

Central to all of these activities, is a strong desire to provide a much more flexible environment for specifying and analyzing higher level requirements and specifications. By providing more rapid and valid feedback at an early stage in a project, vastly superior systems can be developed with substantially less risk.

BACKGROUND:

Historically, the broad effects and high costs of detecting and correcting requirements errors or inconsistencies late in the life cycle have been clearly documented. The earlier their detection the better - thus, there is a very natural desire to detect such errors as early as possible in the life cycle. Early error detection and classification activities focused on analyzing actual program code. Many ideas have been developed which if applied even earlier in the lifecycle could produce substantial savings of time and effort.

REFERENCES:

Wasserman, Tony, UCSF, USE Tool Set

[In particular his rapid prototyping system with its animated execution is quite interesting and contains many directly useful ideas.]

Stucki, Leon, "New Directions in Automated Tools for Improving Software Quality", Current Trends in Programming Methodology, Vol. II, Yeh, R. T. editor, Prentice Hall, 1977.

[Contains interesting ideas for representing and analyzing dynamic properties of systems. The specific examples are aimed at the coding level - but the approaches to dynamic analysis and potential extension to earlier phases of the life cycle are clearly suggested.]

DURATION:

Task 1:	24-36	months
Task 2 [phase I]:	18	months
[phase II]:	36	months

COST:

Tasks 1:	Improved prototyping systems ($750,000 to $1,500,00)	
Task 2 [phase I]:	Feasibility Study & Simple Prototype ($300,000)	
[phase II]:	Initial Operational Analyst Assist ($750,000 to $1,500,000)	

DELIVERABLE:

PRODUCT:	DATE: (months after start)
Task 1:	
Requirements for Prototyping System	4 mos.
Preliminary Design/Test Plan	8 mos.
Detailed Design/Test Procedures	14 mos.
Program Code/Users Manuals	20 mos.
Test Results Summary	26 mos.
Concept Paper for Future Research	36 mos.
Task 2 [Phase I]:	
Requirements for Knowledge-Based System	4 mos.
Preliminary Design/Test Plan	8 mos.
Detailed Design/Test Procedures	12 mos.
Program Code/Users Manuals	16 mos.
Test Results Summary	22 mos.
Concept Paper for Phase II Research	24 mos.
Task 2 [Phase II]:	

Requirements for Knowledge-Based System	4 mos.
Preliminary Design/Test Plan	8 mos.
Detailed Design/Test Procedures	14 mos.
Program Code/Users Manuals	20 mos.
Test Results Summary	26 mos.
Concept Paper for Future Research	36 mos.

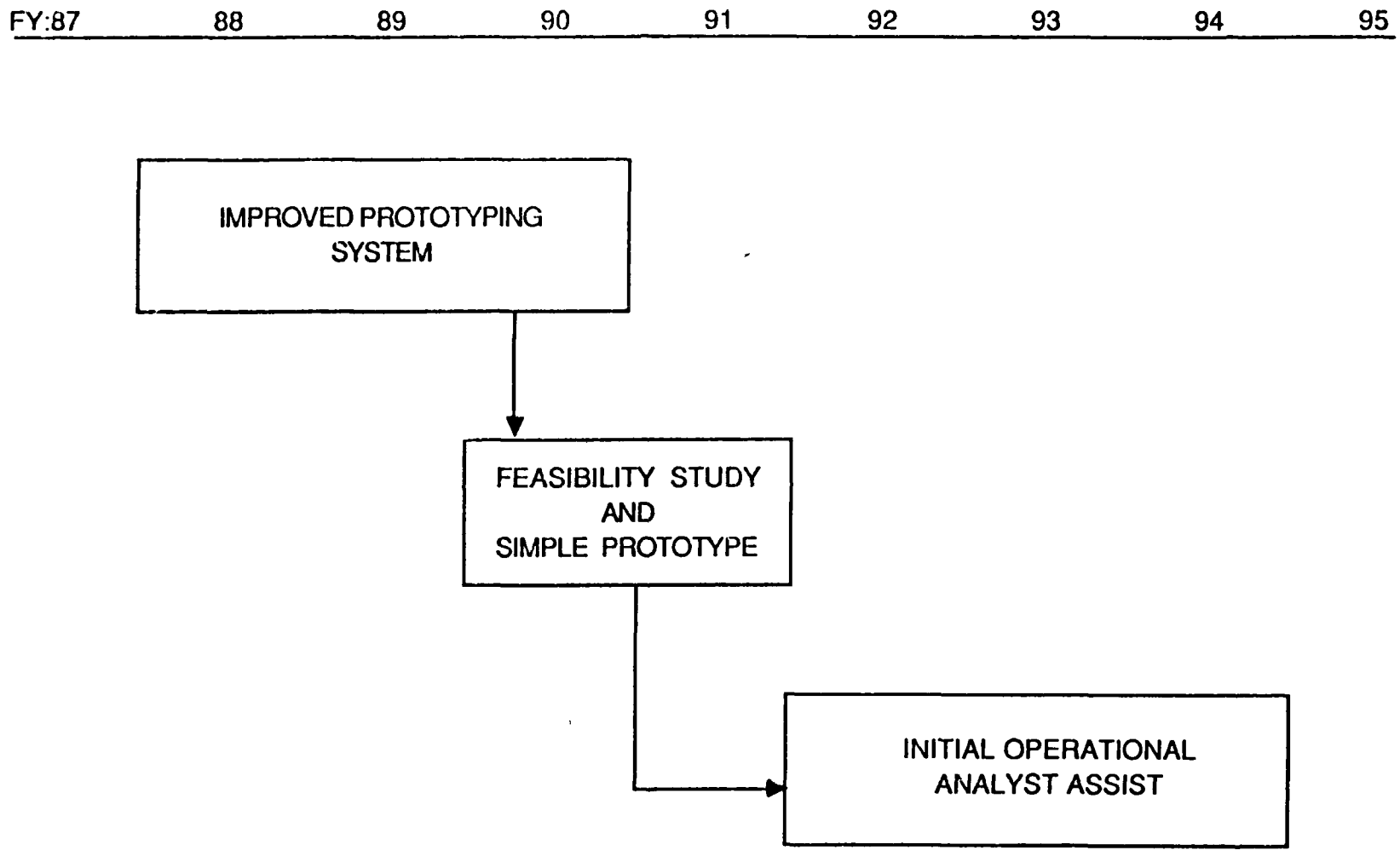


Figure E.1.5-1 DYNAMIC ANALYSIS R&D EFFORT

E.1.6 Solution Architecture Synthesis

RET Research Issue. Evolutionary Track.

PROBLEM DESCRIPTION:

The synthesis activity takes system requirements as input and produces as output a design (solution architecture); both represented as structured text. This involves normal aspects of hardware/software selection/development.

VALUE IN REQUIREMENTS PROCESS:

Synthesis of designs: (1) aids performance analysis, (2) aids examination of functionality alternatives (mission-oriented individuals need to "try it out"), and (3) gives insight into feasibility.

SOLUTION APPROACH: There could be three kinds of assistance:

(1) Expert Advisor: providing tactical guidance on resource selection and management.

(2) Design Critiquer: evaluates the architecture goodness and design style (cohesion, complexity).

(3) The second approach is to provide a specialized design language (VHLL) in which designs would be specified.

Issue (1) implies use of resource models and both (1) and (2) require a fairly formal representation of requirements.

5 YEAR OBJECTIVES:

Enhanced RPS and VHLL Prototyping tools: to aid synthesis: better view/reuse/syntax mechanisms.

10 YEAR OBJECTIVES:

Expert advisor.
Design critiquer.
Specialized design language (VHLL).

RISK:

10-year objectives: Medium/high because of lack of insight.

COST:

Degree of effort required: 5-year objectives: Minimal. 10-year objectives: 2-5 man years for expert advisor, 1-3 man years for design critiquer.

CONTINGENCIES:

10-year objectives: uncertain due to lack of insight.

CONCLUSION:

The "design problem" has not yielded a solution in any CAE field. It is unlikely the limited RET funding can achieve basic results beyond simple extensions to present tools.

AUTHOR NAME:

Michael Konrad, Terry Welch.

E.1.7 Scenario Generation Support & Scenario Coverage Analysis

RET Research Issue. Evolutionary Track.

PROBLEM DESCRIPTION:

Create a scenario that drives the prototype. The scenario may indicate specific actions expected of the prototype for checking results. The scenario should exercise the prototype, yet intelligently react to the prototype's responses (e.g. play the role of an antagonist). Scenario creation should also be sensitive to the type of data required by the prototype.

A coverage analysis tool is needed to indicate the parts of the prototype that were brought into play during its execution, the requirements that are related to each part, and sensitivity of the prototype to scenario input.

VALUE IN REQUIREMENTS PROCESS:

Scenarios play a key role as drivers of prototypes in experiments; often providing the only way to evaluate a system operationally. Tools for scenario creation, coverage, and analysis enhance the experimental usefulness of prototypes.

SOLUTION APPROACH:

Scenario generation and scenario coverage require different technological solutions. Below we examine the issues in this order: (1) generation of adaptive scenarios and driving the prototype, (2) probing the prototype for coverage, (3) given the expected system responses under a scenario, attempt to prove that the requirements and solution architecture satisfy the scenario and derive the coverage, and (4) how these approaches compare to what the Rapid Prototyping System tools (RPS) offer.

(1) GENERATION OF ADAPTIVE SCENARIOS AND HOW TO DRIVE THE PROTOTYPE

Adaptive scenarios are viewed as consisting of two kinds of very high-level descriptions: (1) strategy: what the scenario is trying to accomplish against the (target) system and (2) resources: what resources the strategy has available to carry out its objectives. The corresponding view of prototypes is that there is a prototype environment consisting of (1) the prototype code and (2) a high-level representation of resources (especially sensors and actuators) and two maps: (2a) how system resource

changes (e.g. sensors') map to specific prototype inputs (translation to concrete data and associating the data with prototype code' parameters/ports) and (2b) another map of prototype outputs to system resource changes (e.g. actuators').

Thus before an adaptive scenario can drive the prototype, the following must be specified: (1) scenario strategy and resources, (2) system resources and how they map to prototype inputs/outputs, and (3) how scenario resource actions affect system resources, and vice versa. Then to drive the prototype, all resources are simulated, and the following sequence is repeated:

* the scenario strategy determines what changes to make in the activities of its resources,
* these activity changes drive changes in system sensors,
* sensor changes are mapped into prototype inputs,
* the prototype executes, creating output,
* the prototype's outputs are mapped into actuator changes,
* actuator changes drive changes in scenario resources.

On each successive iteration, the strategy examines how its resources have been affected by system (prototype) action, and appropriately modifies the actions of its own resources to maintain its objectives.

Given the above, how do we support it? We require a high-level simulation system that (1) allows high-level manipulation of objects and (2) provides high-fidelity low-level stimuli to the prototype (and which translates back low-level prototype output into appropriate object changes). Such a system can be partly based on now-available commercial knowledge-based simulation systems, e.g. SimulCraft (Carnegie Group) and SimKit (IntelliCorp). These systems would allow specification of scenario strategy, resources, and resource interactions, and then would simulate all resource activity. However, they do not address item (2) above, i.e. the mappings to/from the prototype. What is needed is a "simulation and code interface", a program capable of interfacing system resource simulations and the prototype, providing: (1) input/output linkages between the two and (2) translation of/to resource changes to/from concrete input/output data.

Under this approach, the simulation system controls the clock, allowing interactive examination of partial results and interactive editing of specific variables during the execution of the simulation.

This addresses the issue of scenario generation and how to drive the prototype.

(2) PROBING THE PROTOTYPE TO DETERMINE COVERAGE

For scenario coverage and sensitivity analysis, a probe is required into the prototype that outputs not only the data that

results from the simulated stimulus but also an indication of the requirements (and/or solution architecture) in terms of data structure, processes and resource called upon to generate that output.

(3) GIVEN SCENARIOS THAT CHARACTERIZE EXPECTED OUTPUT, PROVE THE SCENARIO AND DERIVE COVERAGE

Another approach to the coverage problem is to express the solution architecture (or possibly even the requirements) as a set of PROLOG-like specifications, treat the scenario as a sequence of input/output requirements, and statically check the specifications to see how they can be combined to produce the specified output from the input. The specifications used to prove that the output can be generated from the input will indicate the coverage. This would extend the work of Shapiro (Algorithmic Program Debugging). Note that a prototype is not necessary to determining coverage under this approach, a solution architecture is sufficient.

(4) HOW DOES RPS COMPARE?

RPS. Here we summarize the RPS scenario generation and results analysis capabilities and the premises on which they are based: (1) Scenarios are initially characterized at a military mission level by an operations expert. (2) Later, a functional/process model of the target system is prepared (by another analyst), indicating system stimulus types and the sequence of functions they activate. The model also indicates the resources for which the functions contend. (3) The scenario is iteratively decomposed until it consists of events which are all system stimuli. A scenario library and some generation capabilities help relieve some of the more mechanical aspects of this. (4) The scenario is run against a performance model of the system. (5) During the run, the performance tool collects data, computes measures of how well (time-wise) the modeled system performed, and at the conclusion of the run presents the results in standard statistical fashion. (6) Other possible feedback includes: (for some performance models) a playback-capability (graphical reenactment of each event and the transaction it triggered), a situational display (displaying the viewable aspects of the scenario as it unfolds), and the target system's display (what the target system throws out on screens during transactions triggered by some scenario event). (7) At no time are scenarios run against executable code.

In conclusion, RPS scenarios are canned and the prototype (the system performance model) generates no functional output, only: (1) resource utilization data for the performance tool to accumulate and (2) displays. The motivation behind the RPS approach to scenarios is that such performance data is invaluable.

How would the authors' recommended scenario generation approach extend these RPS capabilities? In the authors' approach, scenarios drive more than just performance models, they drive

executable models which generate functional output that can interact with (and be related to) the unfolding scenario. The result is improved realism in scenario and target system model interactions, yielding insight into how an antagonistic force might overcome the target system and of target system countermeasures. It is important to note that such experiments will suggest revisions in more than just target system performance; functionality and control issues will have to be addressed too. By combining this approach and the capabilities RADC is currently developing in performance models (RPS, to a small degree the Analyst) and executable models (VHLL Prototyping tools), the result is a capability to evaluate target system function, performance, and cost trade-offs.

Thus for maximum effectiveness of the adaptive scenario approach, the authors recommend that the adaptive scenario effort be pursued so that the resulting scenarios can work with models developed under current RADC prototyping efforts.

RPS will provide a capability to represent some types of requirements and solution architecture expressions in a knowledge base for interrogation and analysis of interactions. This capability could be the basis for proving scenarios from system specifications, deriving coverage.

5-YEAR OBJECTIVES:

Provide probes for prototypes.

Provide a knowledge-based simulation baseline system that can interface and run against performance and executable models (especially those constructed by RPS, VHLL prototyping tools, and the Analyst). Build a configuration manager that can semiautomatically generate the simulation/prototype interface (that relates high-level sensor/actuator changes to low-level prototype input/output) based on available sensor/actuator models and user requirements on prototype level of detail and speed.

10-YEAR OBJECTIVES:

Develop a system to statically examine the design of the prototype by analyzing the input/output requirements of a scenario using the design specifications as a proof in a theorem-proving system. Base such a system on RPS knowledge-based requirements/model representation capabilities. The scenario would represent instances of the general theorem represented by the design.

RISK:

The knowledge-based simulation approach to scenario generation and driving the prototype involves little risk, given the commercial availability of what would be a key component of such a system. Both scenario coverage approaches invite risk because tracking what requirements were exercised by a scenario presents

integration and technical risks. Assuming integration risks are controlled, the likelihood of results is 60 percent for prototype probes and 40 percent for the static analysis (theorem proving).

COST:

Knowledge-based simulation	2 man-years
Combine simulation with code	2 man-years
Provide intelligent configuration manager	2 man-years
Prototype probe (dependent on the prototyping approaches)	
Create static scenario coverage analysis system	8 man-years

CONTINGENCIES:

Part of both the scenario generation and scenario coverage research (especially probing) depends heavily on the approaches taken in the prototyping research. Also dependent on the testbed integration effort.

CONCLUSIONS:

This effort should be funded. Recommend that scenarios developed under the scenario generation effort be able to work with all prototyping tools. Recommend static scenario coverage analysis effort be a possible follow-up to the RPS effort.

AUTHOR NAME: Stephen Sherman, Michael Konrad

RET R&D Effort

ISSUE NAME: Scenario Generation Support & Scenario Coverage Analysis

TRACK: Evolutionary

OBJECTIVE:

Produce a knowledge-based simulation tool, a dynamic coverage tool (a prototype probe), and a static coverage tool (proves scenario from system specification).

SCOPE:

The scenario generation effort will not attempt to pioneer knowledge-based simulation. The two coverage tools should be pursued in two parallel R&D efforts. The static coverage tool effort will attempt to pioneer an application of existing technology.

TECHNICAL APPROACH:

Working closely with the prototype development efforts, provide a knowledge-based simulation system for scenario generation. The system must contain a representation of system/scenario resources and the scenario strategy. The system must be extended to include a "simulation and code interface" which translates between high-level system resource changes and prototype inputs and outputs.

For static coverage analysis, the prototype design (solution architecture) is represented as a set of PROLOG-like predicates. The scenario is represented as sequence of input values and output responses. A theorem prover will select the predicates that accept the specified input and determine what helped produce the specified output. The predicates that are required to derive the output from the input are a measure of the coverage. The design predicates must then be related to requirements to indicate requirements coverage. The RPS Knowledge-based system "Translator" can serve as a basis for this effort. The Translator is capable of taking system/operator resource models as input and generating PROLOG-specifications.

BACKGROUND:

The key tools are knowledge-based simulation, knowledge-based analysis and theorem proving.

The concept of knowledge-based simulation was defined by Reddy and Fox of Carnegie-Mellon, "KBS: An Artificial Intelligence Approach to Flexible Simulation", and also implemented in the ROSS simulation system developed by the Rand Corporation, "ROSS: An Object-Oriented Language for Constructing Simulations."

REFERENCES:

1. KBS: An Artificial Intelligence Approach to Flexible Simulation, Y.V.Reddy and Mark S. Fox, CUM-RI-TR-82-1, Robotics Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 15213, 14 September 192.

2. ROSS: An Object-Oriented Language to Constructing Simulations, David McArthur, Philip Klahr, Sanjai Narain, R-3160-Af, December 1984.

3. Algorithmic Program Debugging, Ehud Y. Shapiro, The MIT Press, Cambridge, Massachusetts, London, England.

4. SimulCraft is a commercial product of the Carnegie Group.

5. SimKit is a commercial product of IntelliCorp.

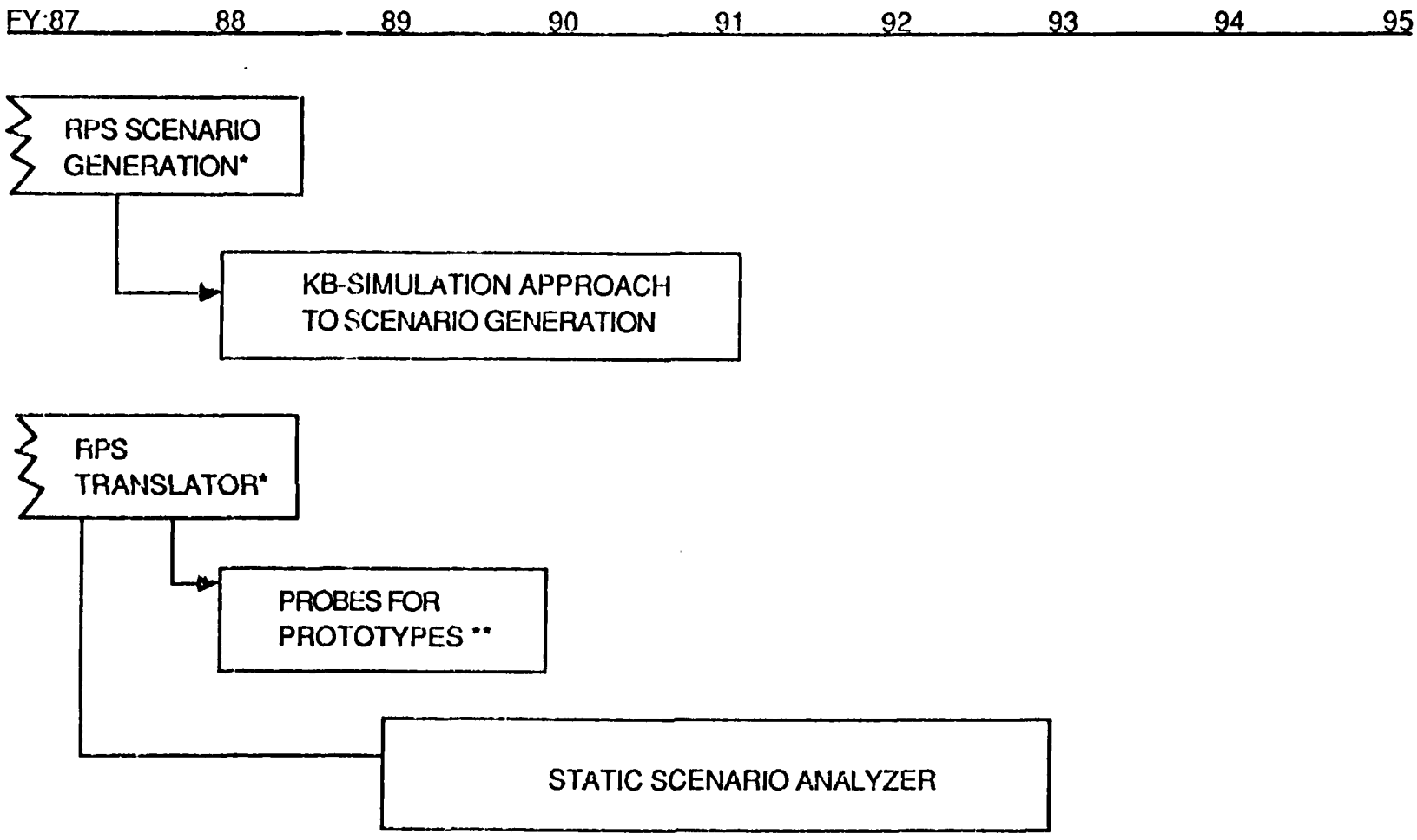
DURATION:

Scenario Generation (SG)	3 Years
Static Scenario Analyzer (SSA)	4 Years

COST:

SG	$ .9 million
SSA	$1.2 million

DELIVERABLE:	Product	Date
SG	Knowledge-Based Simulation	12 months
SG	Simulation and Code Interface	24 months
SG	Configuration Manager-Intelligent	36 months
SSA	Enhance RPS representation of designs	18 months
SSA	Input/Output Representations	24 months
SSA	Theorem Proving Techniques	40 months
SSA	Design-Requirements Connection	48 months



* PART OF CURRENT RPS EFFORT

** PERHAPS A FOLLOW-UP ON EXISTING PROTOTYPING EFFORTS

Figure E.1.7-1 SCENARIO GENERATION SUPPORT & SCENARIO COVERAGE ANALYSIS R&D EFFORT

E.1.8 Validation of Prototype and Scenarios

RET Research Issue. Evolutionary Track.

PROBLEM DESCRIPTION:

There are two problems: (1) Make sure that the prototype and scenario are consistent, complete and logically correct in a static check. (2) Determine the validity of the results of executing a scenario on the prototype in a dynamic check.

VALUE IN REQUIREMENTS PROCESS:

The credibility of the RET depends on (1) our ability to associate requirements with resource requirements and (2) our ability to accurately predict the resource requirements. Thus validation of the prototype and scenario increases confidence in the predictions.

SOLUTION APPROACH:

Provide a knowledge-based syntax and semantics checker for a static check. It is assumed that the prototype and scenario have a formal language description. The knowledge base will also contain meta-knowledge about the domain model. The checker will examine the descriptions for the prototype and scenario for consistency and completeness.

For a dynamic check, the prototype results must be compared with real implementations using the same scenario. The difficulty is in identifying good comparison metrics and in metering both the prototype and implemented system. Unfortunately the results are only useful after decisions based on requirements and prototype accuracy have been made. In order to use information on accuracy, we need a library of validated correct (within limits) prototypes. New prototypes need to be compared to validated prototypes through reusability techniques. If the differences between the new prototypes and the validated prototypes are small, confidence in the accuracy is improved. The key to this approach is the creation of a library of validated prototypes or parts of prototypes and techniques for retrieving information from that library.

5 YEAR OBJECTIVES:

Start a library of validated prototypes and begin developing retrieval techniques. Define a formal language for describing prototypes and scenarios.

10 YEAR OBJECTIVES:

Continue developing the reusable prototype system including breaking down the prototype into reusable, validated prototype parts. Develop the techniques to validate a prototype and scenario for consistency and completeness.

RISK:

The static check is higher risk due to the difficulty of coordinating languages for scenarios and prototypes. However, if that problem is solved, the techniques for determining consistency and completeness being developed for expert systems could be transferred to the domain of prototypes and scenarios.

The risk for the dynamic check is low since reusability is a large research area now and validating our prototypes must be done.

COST:

The static check and language definition is a 6 man year effort. The dynamic check is a continuous effort in building a library of validated prototypes. The effort to convert a reusable programming technique to prototypes is a 3 man year effort.

CONTINGENCIES:

Wait for reusability technology to be developed. Coordinate scenario and prototype formal language descriptions.

CONCLUSION:

Fund the library creation of metered and validated prototypes. Examine a formal language approach for both the prototype and scenario.

AUTHOR NAME:

Stephen Sherman

E.1.9 Scenario Execution and Analysis of Results

RET Research Issue. Evolutionary Track.

PROBLEM DESCRIPTION:

Driving a prototype against a scenario can produce considerable data which must be collected for presentation and analysis. Presentation can occur both during and after the experiment. An analysis capability is needed to evaluate the combined results of many experiments.

VALUE IN REQUIREMENTS ENGINEERING PROCESS:

Providing tools to interpret results of prototype experiments is critical to evaluation of prototypes, and therefore of requirements.

SOLUTION APPROACH:

Two sets of needs are to be addressed, those of a prototype experimenter and those of a mission user. The mission user collaborates with the experimenter in establishing the scope and objectives of an experimental run(s) and in selecting the right scenario(s) for the run(s). The prototype experimenter sets/links up the prototype and scenario for the experimental run(s). During scenario execution, the experimenter may need feedback on how the prototype is doing (e.g. in utilization of resources). The mission user must have feedback in order to assess how the prototype performs against the scenario. The experimenter needs to record/capture comments elicited from the mission user during the experiment. After the experiment, both the mission user and experimenter might want to analyze results of the current run, possibly in the context of previous runs.

Thus the following capabilities should be provided:

(1) Visual presentation and analysis of: the unfolding scenario situation, what outputs (displays, functional responses to the scenario) the prototype generates, and what resources the prototype utilizes. The data presentation format (e.g. of scenario situational displays vs. prototype displays, or what plots against what) should be flexible.

(2) To (1) above should be added an aggregate analysis capability. The experimenter and mission user can specify what kinds of aggregate analyses they want on data generated

during the run, for viewing both during the run and after. Examples of aggregate analyses include: number of resources left (e.g. planes), mean response time, and percentage of hits (e.g. simulated hits on attacking planes). For presentation of aggregate analyses results, standard statistical presentation formats should be available.

(3) A capability for historical analyses of results. For example to do a sensitivity analysis, one might want to do a number of experimental runs in which a scenario is fixed for a time and run against different prototypes and/or vice versa.

(4) A capability to capture mission user responses during/after the experimental runs. This might involve no more than careful recording of what goes on during each experimental run. An additional approach would have the user temporarily pause the experimental run to make "debugging changes" or textual annotations to the unfolding events of what should have been the prototype's response/display/performance.

(5) A capability to formally/informally compare expected prototype outputs (which are part of the scenario's definition) against scenario events. The formal comparison approach is addressed by the static scenario analysis tool recommended as a part of the "Scenario Generation Support & Scenario Coverage Analysis" research issue. For the near-term, the comparison will be done by humans with the assistance of data-management tools.

5-YEAR OBJECTIVES:

A central repository for the collection of all data arising from prototype experimental runs. Database management facilities (e.g. browsing) available to aid human analysis of data.

Specific analyses (especially analyses during experimental runs) remain under the control of the different prototyping tools until they can be integrated.

10-YEAR OBJECTIVES:

From a single experimental run, get a full range of analyses (not just that provided by a single tool on a single model) and in which all data can be cross-correlated (even during the run).

Knowledge-based aids for evaluation of prototype sensitivity. Analyses providing results based on combined experimental information.

A tool that indicates which experimental runs should be performed next based on an evaluation of system requirements that need further testing or stressing.

RISK:

Near-term objectives: low. Long-term objectives involve moderate to high risk.

COST:

Knowledge-based system for analysis of experiment results.	6 man-years.

CONTINGENCIES:

A continuing R&D effort in prototyping. Reasonably successful integration of the RET, especially in the area of the database. Success in the "adaptive scenario generation" approach of the "Scenario Generation Support & Scenario Coverage Analysis" RET research issue.

CONCLUSION:

Recommend funding. It is crucial to the RET. No special funding is needed for the near term, provided that (1) the "Database" RET development effort provides the appropriate data management facilities and (2) all prototyping tools provide the appropriate capabilities for the analysis of scenario-prototype execution results.

AUTHOR NAME:

Stephen Sherman, Michael Konrad.

RET R&D Effort

ISSUE NAME: Scenario Execution and Analysis of Results

OBJECTIVE:

Provide: visual presentation and analysis, aggregate analysis, and historical analysis of results from driving a prototype against a scenario.

SCOPE:

Research addresses the following subjects: statistical analysis of data, control of prototype experiments, sensitivity analysis, and experiment design.

TECHNICAL APPROACH:

Near-term approach:

RPS. In the "Scenario Generation Support & Scenario Coverage Analysis" research issue, the capabilities of RPS' performance modeling tools is discussed. These tools work off performance models, collect performance data, and provide relevant analyses on the results. Thus RPS provides significant capabilities in visual presentation and analysis (items (1) and (2) of the Solution Approach) for performance data.

RET database. The RET database should serve as the collector and repository for all information generated during prototype runs. This would help, for example, compare results of running the same scenario against both a performance model (RPS) and executable model (VHLL Prototyping tools) as a basis for a trade-off analysis. The RET database should provide data management support for human analysis. These requirements on the RET database are consistent with those described under the "RET Database Management System" R&D effort description.

However, there is still a need to provide specific analyses when scenarios are run against specific target system models, e.g. the different analyses that are performed on the different performance models one can construct with RPS tools. For the near term, these analyses will probably remain under control of the tool that helped construct the model/prototype. With testbed integration, the experimenter should be able to orchestrate the running of a scenario against several models (say, of the same solution architecture) and combine analyses.

Long-term approach:

Expand the adaptive scenario generation capability discussed in the "Scenario Generation Support & Scenario Coverage Analysis" RET research issue so that the simulation knowledge base can be used

to evaluate prototype sensitivity, e.g. between different scenarios and different prototypes. Analyses should provide results based on combined experimental information.

Assuming that requirements, scenarios, and prototypes are maintained in the same knowledge base (of which the simulation knowledge base constitutes a part), another capability can be identified for the very long term. From an aggregate analysis of sensitivity/coverage analyses results, an indication will be provided of what future experimental runs should be performed based on an evaluation of system requirements that need further testing or stressing.

BACKGROUND:

The authors know of no mature efforts on analyzing results from prototype runs, though there are analogs in testing technology. However, [1] examines the overall software prototyping approach, investigates the experimenter-mission user relationship, and makes some practical suggestions of how to make the prototyping experiment effective.

REFERENCES:

[1] Alavi, Maryam, "An Assessment of the Prototyping Approach to Information Systems Development", Communications of the ACM, Vol. 27, No. 6, June 1984.

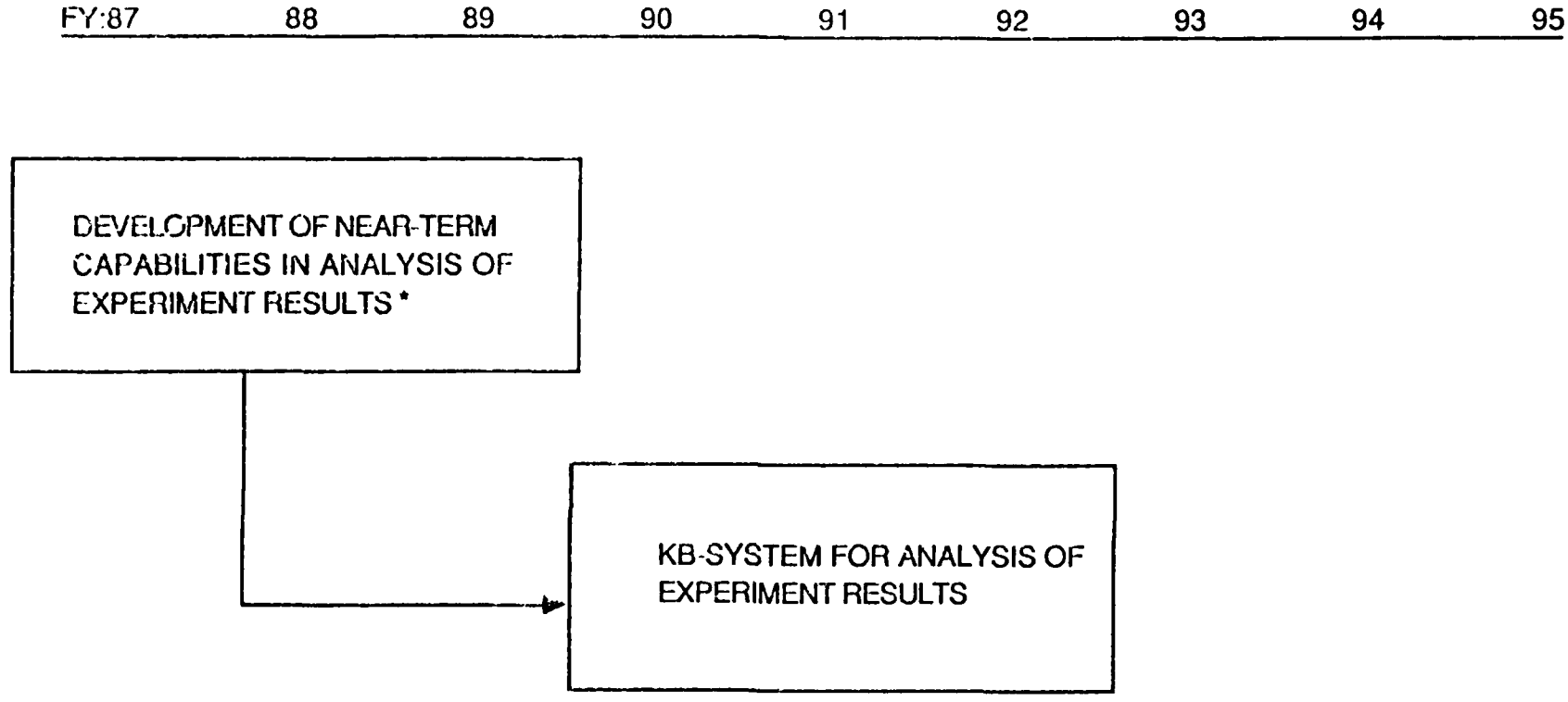
DURATION:

Knowledge-based system for analysis of experiment results (AER)	3 years

COST (in $):

AER	$ .9 million

DELIVERABLES: Product	Date (months after start):
Enhance simulation knowledge base/RET database for Historical analyses	16 months
Prototype sensitivity analysis tool	24 months
Experiment advisor tool	36 months



* POSSIBLY NO SPECIAL FUNDING REQUIRED

Figure E.1.9-1 SCENARIO EXECUTION AND ANALYSIS OF RESULTS R&D EFFORT

E.1.10 Estimation of cost, risk, time in system development, Performance & Execution costs analysis

RET Research Issue. Evolutionary Track.

PROBLEM DESCRIPTION:

The uncertainty of software costs is one of the major cost issues. The existing cost models have several subjective parameters. The manager dealing with these things needs tools for more reliable estimations of cost, time etc. to aid making scheduling and resource allocation decisions.

VALUE IN REQUIREMENT PROCESS:

Better estimates of cost, time and performance can help management in choosing between alternate solutions. This also facilitates the examination of functionality vs. cost trade-off (also known as impact analysis or sensitivity analysis) for better project management.

SURVEY OF CURRENT TECHNIQUES:

Cause-and-effect relationships as a basis for software cost analysis and estimation have been investigated by various people for the last twenty years. Most of the models use the size of the project to estimate the man-months required to complete the project, as this factor seems to correlate best with the cost. These models are based on data from past projects. Sometimes the data is translated to charts and graphs. Another approach is to formulate a parametric model, involving mathematical functions of several variables, suggested by previous experimentation and engineering judgment. Statistical techniques are used to determine the relevance of the set of variables used in the equation; constants of the equation (parameter estimation) are based on historical data.

[Wolverton 75] has classified these efforts into five categories -- Top-down estimation, Bottom-up estimation, similarities and differences estimation, ratio estimation and standard estimation. [Boehm 81] has a slightly different classification of these approaches -- based on algorithmic modeling, expert judgment, estimation by analogy, pricing to win (cost = f(what customer can pay)) and Parkinson's law ("The project expands to consume the budget available"). Both of them observe that two of these models should be used for better reliability.

Besides estimating the total cost of development of the entire system, it is useful to have separate estimates for each phase of the life cycle of the system. Similarly estimates for each module and unit may be needed to optimize the schedule and cost against deadlines penalties.

Past success stories include Wolverton's studies (of 1974) and the RCA PRICE S system [Friedman 79]. Recent success stories include the Walston and Felix studies (IBM) and COCOMO model [Boehm 81].

The reliability research has two distinct flavors. The first one is oriented towards tools for static code analysis, test case generation, symbolic execution, correctness proofs etc. The other is nearer to the statistical modeling approach and leads to reliability models, methodology for code inspection, standard test procedures etc. The models for measurement, estimation and prediction are based on a variety of estimation statistics (Rayleigh, Baysian, Markov, Geometric etc) for failure rates/error distribution. Recent methods of fault handling by fault-tree, event-tree, and influence-tree analysis also seem to be promising methods. The Spiral model of the software life cycle has risk analysis in every phase. Success stories include IBM Clean Room and Bell Labs ESS.

The modeling approach has its own limitations. Data from old projects may not be useful for estimating the cost, risk etc. of a new project. Rapidly evolving technology (better programming environments, richer set of tools, automatic programming, workstations) have changed the productivity of programmers. The models can not take many factors (such as implementation constraints, management techniques, programmer qualification) into account. Solutions based on concurrent programs and distributed systems pose problem for these models. The models should be supplemented by other informal means of estimation of cost, time and risk of development.

SOLUTION APPROACH (1): METRICS

A metric is a measure of some characteristics of a software system. In the Metrics Guided Methodology [RAM 85], metrics are used as a tool for software development. The metrics provide feedback to the developer, letting him know his progress. It can also be used predict where the project is going by estimating future size and cost, or it may indicate that the current design is too complicated and unstructured. It can be used through the software life cycle from predictions/estimations about new products to evaluation/ maintenance of existing products.

The Metrics Guided Methodology (MGM) helps get better estimates of the cost, time and performance of the system being designed. Requirements metrics can indicate the complexity of designs and implementations at an early stage and suggest simplifications in particular areas or allocation of more resources in those areas.

These can help in making intelligent compromises between target system performance, project deadlines and allocation of manpower to the project [RAM 85].

The use of metrics for prediction of system quality/difficulty of design etc. can save a lot of effort and investment in prototyping. The metrics substitute partially for prototyping of the system. "Feedback metrics" give immediate feedback to the designer and hence he does not have to wait until later design stages to detect problems. This leads to smaller and fewer design iterations for the same quality. The measurement and interpretation of metrics can be automated by expert system technology.

When using MGM for a specific project, metrics should be collected based on the objectives we want to achieve in that project. This view has been expressed by several people [BAS 84]. This requires one to obtain the objectives (e.g. system availability, performance) from the requirements and then develop/select metrics for those objectives. Development of metrics is mostly subjective at present, though limited validation can be done by experimental means. Using prior data and carefully documented results from past analysis on prototypes or similar projects can also be used to boost confidence in particular metrics chosen for the project.

BACKGROUND:

The term "metrics" have been traditionally used to denote complexity measures. Metrics have also been used for maintenance of large software systems. The term metrics, in general, refers to all kinds of measures of various characteristics of software. The particular characteristics may be useful for assessment and estimation of the quality of requirements and prediction of difficulties in later stages of software lifecycle. Other measures include the use of techniques like utility functions and risk analysis in making decisions during software development.

Metrics are used to evaluate software process and product. They are also used as a tool for software development. They can be used to monitor the stability and quality of existing systems. There are several classifications of metrics [Basili]. Metrics can be divided into product metrics (number of decisions and interfaces) and process metrics (based on time of development, number of errors). A second dichotomy is quality metrics (which would evaluate the product as good or bad relative to a specific model) vs. invariant metrics (which are independent of environment and product except for the effects of size on cost). A third classification would distinguish between a priori metrics and a posterior metrics. An a priori metric is used to estimate and evaluate the product being designed. An a posterior metric is a measure of the existing product after the design is complete.

It is suggested that a single metric is not sufficient for large software projects and hence a spectrum of metrics should be used

[RAM 85]. Metrics to be used should be selected based on the objectives of a particular project. There is a need for experimental validation of metrics, but little validation can be expected because of the difficulty in conducting controlled experiments in this area.

RESEARCH ISSUES:

There is a need to define and validate a set of metrics to estimate the cost, risk and time required in each phase of development of a software system. A typical validation process can be used [Boehm 78], [McCall 77]. The selection of metrics is subjective and based on experience. Then one clusters the metrics by factor analysis. This is followed by collection of data and regression analysis. The metric scoring method is an important issue, and the method should be defined properly and then automated.

The "size" of the project seems to a major factor in determining the cost of the project. Current metrics used in size estimation (SLOC, number of modules etc) are not very satisfactory. It relies heavily on the ability of the software engineer in-charge to estimate the size of software product based on the requirements, which may be imprecise or subject to change. There is a need for tools to estimate the size of the system from the requirements, using knowledge about past projects and current technology. Similarly, metrics that represent project size (SLOC, number of modules etc) are not satisfactory for many purposes. There is a need for better metrics to represent the magnitude of the project.

The collection of metric data from old projects can be used to calibrate estimation models so that these models can be applied to new projects. Similarly, design and validation of metrics that measure the effects of programming environments, project constraints etc. would be useful. There is definitely a need for developing methods for estimating the cost, time and risk involved in developing distributed systems, concurrent programs, knowledge based systems etc. A methodology to derive a good set of metrics for these purposes has been discussed in [Boehm 78] and [McCall 77].

REFERENCES:

1. Basili V. and Weiss D.M., "A Methodology for collecting valid Software Engineering data", IEEE Transaction on Software Engineering, Vol. SE-10, Nov. 1984.

2. Boehm B.W., Software Engineering Economics, Prentice Hall 1981.

3. Boehm B.W., "A Spiral Model of Software Development and Enhancement".

4. Boehm B.W., Brown J.R., Lipow M., "Characteristics of Software Quality", New York: North Holland, 1978.

5. Friedman F.R., et al, "Price Software Model - Overview", Internal paper, Price system, RCA Corporation, Cherry Hill, NJ, 1979.

6. Henry S., Kafura D., Harris K., "On the relationship among three software metrics", Performance Evaluation Review, Vol. 10 Number 1, 1981 pp. 3-10.

7. McCabe T.J., Young L.F. et al, "Design Basis Paths : A complexity Driven Design Inspection Methodology", Proceedings of Total System Reliability Symposium, Gaithersburg, MD December 1983.

8. McCall J.A., et al., "Factors in Software Quality", Tech. rep. 77c1s62 Command and Information Systems, 1977, Sunnyvale, CA : General Electric.

9. Ramamoorthy C.V., So S.S. "Software Requirements and Specifications: Status and Perspectives".

10. Ramamoorthy C.V., Usuda Y., Tsai W. and Prakash A., "GENESIS : An Integrated Environment for supporting Development and Evolution of Software", COMPSAC 1985.

11. Ramamoorthy C.V., Tsai W.T., Yamaura T. and Bhide A., "Metric Guided Methodology", COMPSAC 1985.

12. Walston C.E., Felix C.L., "A Method of Programming Measurement and Estimation", IBM Systems Journal Vol. 16, No. 1, 1977.

COST:

(1)	Metric Guided Methodology metrics for cost, time, risk; project size metrics. Measurement/analysis tools	3 person yrs	.45 million
(2)	Add Metrics for distributed systems, etc.	2 person yrs	.3 million
(3)	Tools for fault-handling analysis and reliability estimation	3 person yrs	.45 million

DELIVERABLE PRODUCT:

1. Metrics for estimation of "size" of the projects.

2. Methodology for estimation of cost, time and risk of a project. This will combine rapid prototyping and predictive

metrics for faster and economical estimation. New metrics and Tools for measurement and analysis of the metrics.

3. Metrics and tools for estimation of cost, time etc. of projects in new areas of Distributed Systems, Knowledge-based Systems etc.

4. Tools for fault tree, Event tree and Influence tree analysis for fault handling and estimates of reliability from system structure.

RISK:

The metrics are often subjective and are defined on the basis of experience. There should be an attempt to develop an acceptability model to make the metrics acceptable. Experimental validation and peer review should be used for wider acceptability.

AUTHOR NAME:

C.V. Ramamoorthy

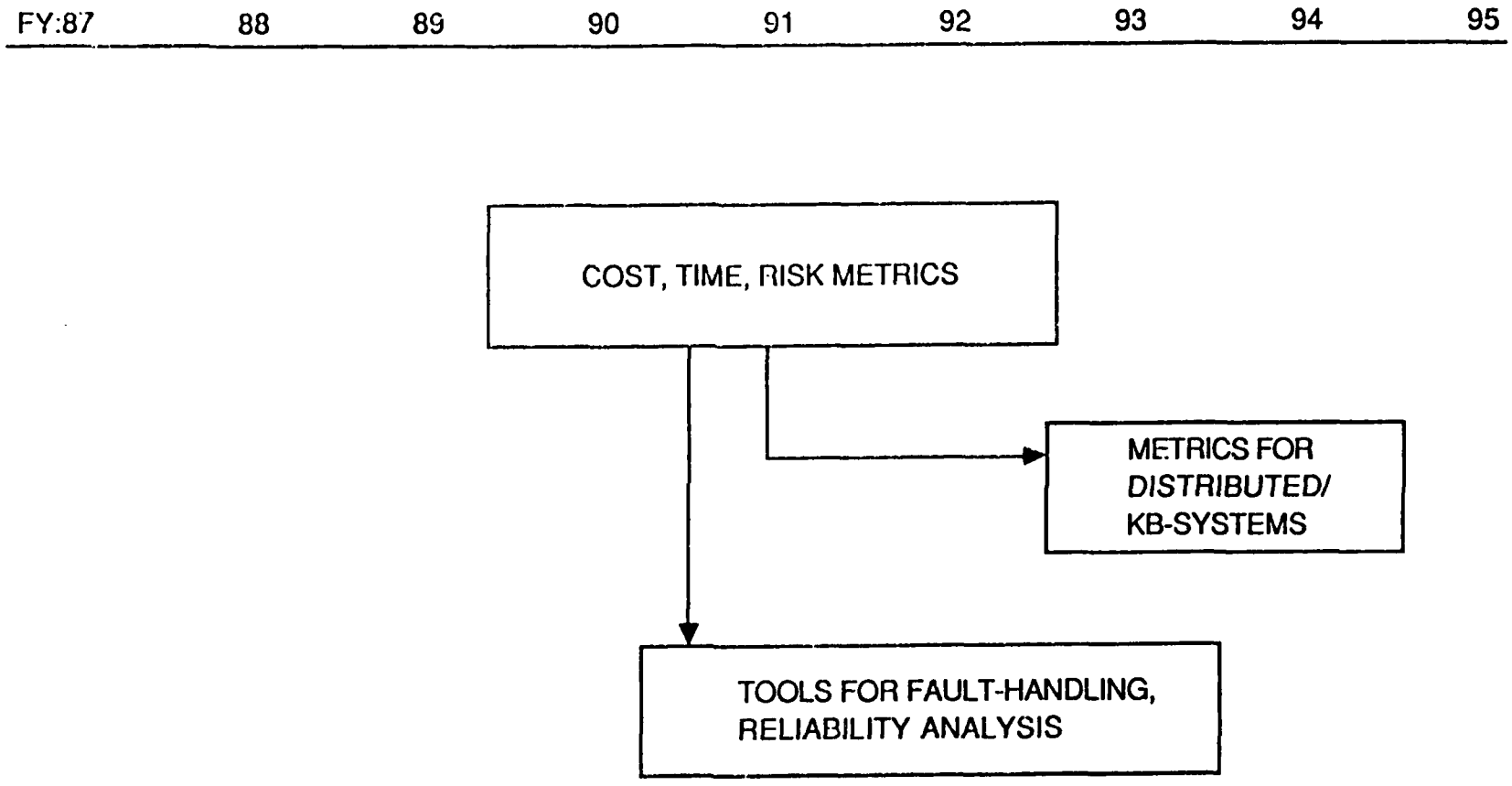


Figure E.1.10-1 ESTIMATION OF COST, RISK, TIME IN SYSTEM DEVELOPMENT, PERFORMANCE & EXECUTION COSTS ANALYSIS R&D EFFORT

E.1.11 Requirements Evaluation

RET Research Issue. Evolutionary Track.

PROBLEM DESCRIPTION:

Requirements are informal statements of need. Requirements bound the space of possible specifications. They should be internally consistent and feasible within the state of the art. The requirements affect the rest of the design process tremendously and any mistake, missing information, or inconsistency may be very difficult to correct at a later stage of design. Similarly the quality of requirements affect the complexity of work in later stages. Hence it is important to create quality requirements and there should be tools to measure the quality of the requirements.

VALUE IN REQUIREMENT PROCESS:

Requirements evaluation will help one to choose between two alternate sets of requirements for the same system. Evaluation can expose statements that complicate later design stages and suggest restructuring to improve them. Measurements of ease of modifiability will help one to make the requirements more maintainable.

Indications of when to terminate iterative requirements engineering activities are needed. Though it is difficult to measure, some estimate of sensitivity to missing knowledge is required if one is to have confidence in the system.

SURVEY OF PAST APPROACHES:

Requirements evaluation is a relatively new area. Requirements are often informal and evaluated by inspection. The criteria for evaluation are still evolving. Heninger (1980) suggested that: requirements should be modifiable, used as reference for maintenance, reflect forethought about the life cycle of the system, and characterize acceptable response to undesired events. He also suggested a format for the requirement definition document.

The first step in requirements definition is to produce a conceptual model of the system. There exists many different kinds of formal models whose use reduces ambiguity and vagueness in the requirements. Objectives models help to hierarchically analyze and describe customer goals in graphic or text form. Conceptual data models help in analysis of major data and their

relationships, helping the capture and structuring of information needs. Conceptual process models help in the analysis of those target system processes identified in the objectives model, and also help analyze process-process and process-environment interactions. Data flow and control flow models help to describe the behavior of the system in more detail [Miyamoto, Yeh].

The requirements are sometimes expressed in natural language or using graphic symbols. Some work has been done to structure and format natural language, without imposing rigorous syntax or semantics. Formal languages are better suited to requirements evaluation, but the use of formal languages for stating requirements has not lead to any great success. Natural language expressions of requirements are difficult to check for completeness and consistency. It is also difficult to partition such expressions into different types (e.g. functional, non-functional requirements), unless the specifier was very careful. They tend to be ambiguous, unclear and inconsistent. Most practical systems fall into the second category: structured natural language representations. These include PSL/PSA, SADT and RSL [see references]. There have also been attempts to use an Ada-like notation, but their use often leads to expression of many low-level details, unnecessarily limiting the freedom of designers later in development.

Requirements are often divided into two types -- functional and non-functional. Functional requirements identify the system services wanted by the user. Often these don't relate (are orthogonal) to the implementation. In principle, the functional requirements of a system should be both complete and consistent. Completeness means that all services wanted by the users should be specified. Consistent means that no two requirements should contradict each other. The non-functional requirements include constraints on the implementation: response time, time of completion, compatibility with existing software and hardware, etc. These often change as technology changes. They often tend to conflict with the functional requirements and induce tradeoffs in the design. The non-functional requirements are generally expressed in natural language.

The purpose of requirement validation is to check for the consistency, completeness and feasibility of requirements. One tool to assist the human validator would retrieve the set of all requirements referencing a common function of the system. Simulation is often used to show the feasibility and completeness of the requirements. Simulation can be often very costly and time consuming. Furthermore it is difficult to change the simulator, as the requirements change and evolve [Vick, Davis 1977]. Rapid prototyping has been tried. Rapid prototyping can be accomplished thru use of high-level languages, libraries of utilities (c-shell in Unix), and/or by reducing the error-handling and quality of the user interface.

SOLUTION APPROACH (1): METRICS

A metric is a measure of some characteristics of a software system. In the Metrics Guided Methodology [Ram 85], metrics are used as a tool for software development. Metrics provide feedback to the developer, letting him know his progress. It can also be used to predict where the project is going by estimating future size and cost, or it may indicate that the current design is too complicated and unstructured. It can be used throughout the software life cycle from predictions/estimations about new products to evaluation/maintenance of existing products. For example, the Spiral model of the software life cycle uses risk analysis at every stage. Metrics can be used in every stage to estimate the risk factors.

The Metrics Guided Methodology (MGM) helps get better estimates of the cost, time and performance of the system being designed. Requirements metrics can indicate the complexity of designs and implementations at an early stage, and suggest simplifications in particular areas or allocation of more resources in those areas. These can help in making intelligent compromises between target system performance, project deadlines, and allocation of manpower to the project [RAM 85].

BACKGROUND:

Metrics are used to evaluate software process and product. They are also used as a tool for software development. They can be used to monitor the stability and quality of existing systems. There are several classifications of metrics [Basili]. Metrics can be divided into product metrics (number of decisions and interfaces) and process metrics (based on time of development, number of errors). A second dichotomy is quality metrics (which would evaluate the product as good or bad relative to a specific model) vs. invariant metrics (which are independent of environment and product except for the effects of size on cost).

A third classification would distinguish between a priori metrics and a posterior metrics. An a priori metric is used to estimate and evaluate the product being designed. An a posterior metric is a measure of the existing product after the design is complete.

It is suggested that a single metric is not sufficient for large software projects and hence a spectrum of metrics should be used [RAM 85]. Metrics to be used should be selected based on the objectives of a particular project. There is a need for experimental validation of metrics, but little validation can be expected because of the difficulty in conducting controlled experiments in this area.

RESEARCH ISSUES:

Normally a designer would like to use several representation schemes for the design. The various representations should be checked for consistency, i.e. whether they could represent the

same system. Tools should be developed that provide graphic and textual representations based on several models, and keep the different representations consistent.

The human expert needs tools to check for the consistency of the requirements. Tools to be developed include browsers for browsing requirements and consistency-checking theorem provers that analyze the system being designed.

Prototypes are useful for requirement evaluation. We would like to explore how the predictive approach (metrics) can help rapid prototyping by reducing the detail and amount of implementation.

There is a need to define a set of metrics to estimate design understandability, modifiability and complexity from the requirements. A typical validation process can be used [Boehm 78], [McCall 77]. The selection of metrics is subjective and based on experience. Then one clusters the metrics by factor analysis. This is followed by collection of data and regression analysis (the "well cycle" involves: building models, taking measurement, and performing analysis to validate the models.) A formal model of requirements evaluation would make things more concrete by providing criteria for goodness of the requirements. Metrics can be defined on the basis of that model. The metric scoring method is an important issue, and the method should be defined properly and then automated.

Test cases can be generated from the requirements. This can be achieved through transformation or via expert system guidance.

REFERENCES:

1. Basili V. and Weiss D.M., "A Methodology for collecting valid Software Engineering data", IEEE Transaction on Software Engineering Vol. SE-10, Nov. 1984.

2. Beck L.L., Perkins T.E., "A Survey of Software Engineering Practice: Tools, Methods and Results", IEEE Trans. Software Eng. SE-9, 1983.

3. Bell et al, "An Extendable Approach to Computer-Aided Software Requirement Engineering", IEEE Trans. Software Eng, SE-3(1), 1977.

4. Boehm B.W., Software Engineering Economics, Prentice Hall 1981.

5. Boehm B.W., "A Spiral Model of Software Development and Enhancement".

6. Boehm B.W., Brown J.R., Lipow M., "Characteristics of Software Quality", New York: North Holland, 1978.

7. Heninger K.L., "Specifying Software Requirement for Computer System: New Techniques and their Applications", IEEE Trans. Software Eng SE6(1), 1980.

8. Henry S., Kafura D., Harris K., "On the relationship among three software metrics", Performance Evaluation Review, Vol. 10 Number 1, 1981 pp3-10.

9. McCabe T.J., Young L.F. et al, "Design Basis Paths: A Complexity Driven Design Inspection Methodology", Proceedings of Total System Reliability Symposium, Gaithersburg, MD December 1983.

10. McCall J.A., et al. "Factors in Software Quality", Tech. rep. 77cis62 Sunnyvale, CA: General Electric; Command and Information Systems, 1977.

11. Miyamoto, Yeh R.T., "A Software Requirement and Definition Methodology for Business Data Processing", Proc. 1981 NCC AFIPS Press 1981.

12. Ramamoorthy C.V., Tsai W.T., Yamaura T. and Bhide A., "Metric Guided Methodology", COMPSAC 1985.

13. Ramamoorthy C.V., So S.S. "Software Requirements and Specifications: Status and Perspectives".

14. Ramamoorthy C.V., Usuda Y., Tsai W. and Prakash A., "GENESIS: An Integrated Environment for supporting Development and Evolution of Software", COMPSAC 1985.

15. Schoman and Ross, "Structured Analysis for Requirement Definition".

16. Teichrow, Hershey "PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Info Processing Systems."

17. Vick C.R., Davis C.G., "The Software Development System".

Last three references from IEEE Trans. Software Engineering SE3-(1), 1977.

SOLUTION APPROACH (2): CRITIQUER VIA KNOWLEDGE-BASE/DOMAIN KNOWLEDGE

The purpose of a critiquer is two-fold. (1) It is a flexible way to measure the quality of requirements expressions (understandability, clarity, consistency) and requirements functionality (modifiability, evaluation and comparison of alternatives, complexity and feasibility). (2) The critiquer can be used to automate the measurement, analysis, and interpretation of other metrics. This is a fast and reliable way to deal with the classical metrics.

A characterization of the kinds of errors that can be present in the requirements would include: errors of inconsistency/incompatibility, errors of quality (understandability, over-constraintment, redundancy, risks to cost/schedule etc), and errors of incompleteness [Bell 76].

Consistency of the requirements can be checked with the assistance of a theorem-proving tool. By adding enough knowledge about the domain we can make the theorem prover a better assistant. Quality can be measured by defining appropriate metrics on the software. Still, it is impossible for a person to measure these properties for a very large software system. Similarly the interpretation of a spectrum of metrics' values will be boring and difficult for a person.

BACKGROUND:

Rule-based Expert Systems is an established technology now. This technology has provided techniques for dealing with inherently ill-defined, difficult, large and complex problems. Traditionally, it has taken a long time for one to gain enough experience to become an expert in these areas. Knowledge in these areas tends to be inexact, evolving, and difficult to formalize. The rule-based programming paradigm scores will over the traditional programming paradigm (e.g. Lisp-based systems vs. traditional Pascal environments).

The power of the rule-based paradigm comes from separation of knowledge and reasoning. This makes it easier to add knowledge about the domain incrementally. It also facilitates quick experimentation and modification of such rule-based systems. Rules make systems such as the critiquer adaptable and tailorable to any project. Different models of software development (Waterfall, Spiral with rapid prototyping) can be used just by changing the set of rules. Rules are also helpful in automating certain tasks, as they become well understood. Rules can help maintain standards for uniformity and quality [Ram 85].

Research in Artificial Intelligence has provided several knowledge representation schemes and an inference engine technology to use that knowledge. There are tools available for knowledge acquisition (i.e. the transfer of problem-solving expertise from an expert to the program).

RESEARCH ISSUES:

The first solution approach recommends that a theorem prover be used to assist designers in checking for consistency of the requirements. Simple tools of this kind can help locate requirements which potentially give rise to conflicts. When added to the knowledge acquisition system, the resulting system would learn about the system being designed as well as assist in detection of inconsistencies. It could generate test cases for checking the completeness of the final design. The knowledge

acquisition capability helps by shifting different kinds of checking to the machine, as they become routine and better understood.

It is widely recognized that software development is a knowledge-intensive process. The seemingly inherent shortcomings of current approaches to software development, demonstrate our limited understanding of the process and product involved [Arrango 85]. There is a need to explore the nature of these processes and develop explicit representations, so that we can reason about them.

One should separate software engineering knowledge from domain knowledge because their application is relatively orthogonal. One should also create good representation and manipulation schemes for these.

There should be some experimental work to develop a set of rules to criticize the properties of the requirements and give feedback to the user. To manipulate the metrics one has to design suitable rules for measurement and interpretation of data. The rules will need experimental validation. The development of criteria of the goodness of requirements is also needed. These developments are complementary to the modeling of software engineering knowledge.

REFERENCES:

1. Arrango G., Freeman P., "Modeling knowledge for software development" Third International Workshop on Software Specification and Design 1985, IEEE Computer Society.

2. Bell T.E., Thayer T.A., "Software Requirements: Are they really a Problem", Proceeding 2nd International Conference on Software Engineering 1976 IEEE.

3. Buchanan B.G., Feigenbaum E.A., "DENDRAL and Meta-DENDRAL", Artificial Intelligence 11:1 (1978) pp.5-24.

4. Clark K.L., McCabe F.G., "PROLOG: a language for implementing Expert Systems", Machine Intelligence.

5. C.L. Foggy, "The OPS-5 User's Manual" Technical Report, Carnegie Mellon University 1980.

6. Greenspan S. "Requirement Modeling: A Knowledge Representation Approach to Software Requirements Definition" CSRG, U. of Toronto, Tech. Rep. CSRG-155, March 1984.

7. Leung C.H.C., Choo Q.H. "A Knowledge-base for effective Software Specification and Maintenance", Third International Workshop on Software Specification and Design 1985, IEEE Computer Society.

8. Dana S. Nau, "Expert Computer Systems", Computer Feb. 1983, pp. 63-85.

9. Ramamoorthy C.V., Garg V. and Aggarwal R. "Environment Modeling and Activity Management in GENESIS", 2nd conference on software tools, techniques and development 1985.

COST:

(1) Metric Guided Methodology - and metrics, and tools	3 person yrs	.45 million
(2) Knowledge-base manipulation of design knowledge	3 person yrs	.45 million
(3) Correlate different design representations	1 person yrs	.15 million
(4) Predictive metrics and prototyping	1 person yrs	.15 million

DELIVERABLE PRODUCT:

1. Expert System Tools for knowledge acquisition and manipulation of knowledge about the system being designed, for locating potentially conflicting requirements, for critiquing the understandability, modifiability and feasibility of requirements.

2. Metrics for measuring the understandability, modifiability, feasibility of requirements, and complexity of later stages of design. Tools for measuring the metrics and interpreting them.

3. Tools to support several different representation schemes for the same design. These would maintain version and configuration information and try to keep different representation for the same design consistent.

4. Methodology to combine predictive metrics with prototyping for rapid prototyping.

RISK:

The tools would assist a designer. Total automation is not being attempted since that does not seem feasible. There should be an attempt to make the tools general yet tailorable to a particular design environment. This added level of abstraction increases the risk.

AUTHOR NAME:

C.V. Ramamoorthy

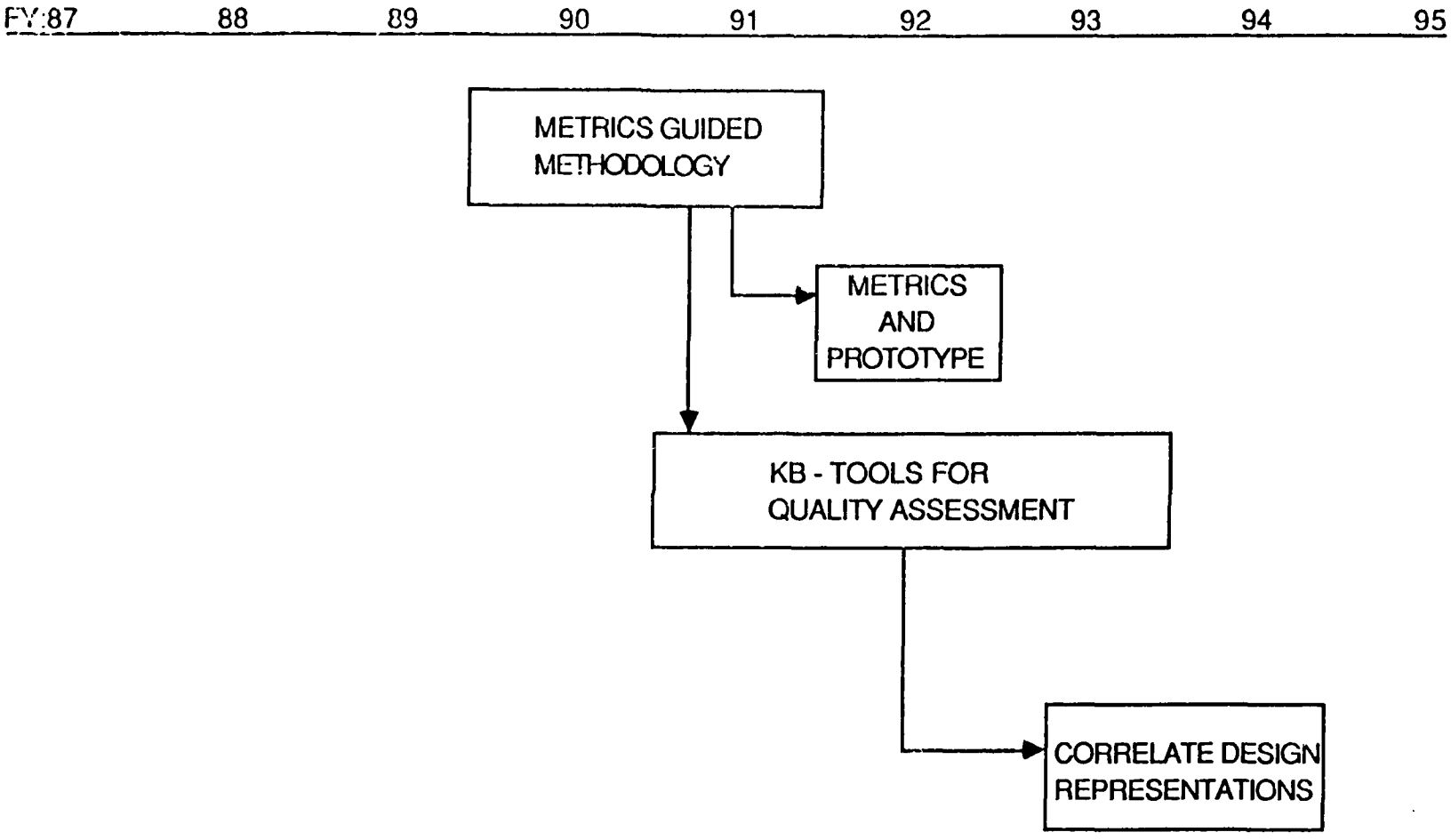


Figure E.1.11-1 REQUIREMENTS EVALUATION R&D EFFORT

E.1.12 Testbed Effectiveness

RET Research Issue. Evolutionary Track.

PROBLEM DESCRIPTION:

New tools and techniques for requirements engineering will be developed because of the early leverage they bring to requirements problems. To ensure that the Air Force employs the best tools and techniques, their effectiveness in producing correct requirements must be determined. This is a key purpose of the requirements engineering testbed.

VALUE IN REQUIREMENTS PROCESS:

Demonstrating that use of a tool leads to improved requirements and better productivity will assist its adoption by the Air Force community. Use of proven tools will lead to better requirements.

SOLUTION APPROACH:

The following things will be measured for the indicated characteristics:

(1) all tools: amount of user time/effort and testbed resources spent, number of requirements errors caught,

(2) prototyping (building prototypes to determine how best to adjust requirements to cut cost/risk, and improve schedule): estimated savings in cost, risk, schedule,

(3) the entire requirements engineering process: cumulative effort and testbed resources, number of errors caught (not cumulative, but before/after comparison), estimated savings in cost, risk, schedule, and

(4) sensitivity analysis: same as 3.

Measurements of the improvement in requirements in 3 and 4 will be proportionately allocated back to utilized tools according to user effort spent. Before/after comparison of requirements will be a largely manual effort assisted by general documentation/versioning tools.

5-YEAR OBJECTIVES:

Testbed instrumented for effort and resource measurements.
Prototyping providing basis to cost, risk, and schedule estimates.

10-YEAR OBJECTIVES:

Cost, risk, schedule estimation capability.
User-preference measurement of tools at end of process.

RISK:

It will be hard to evaluate the effectiveness of individual tools; easier to evaluate the effectiveness of the entire process. 5-year and 10-year objectives: significant risk in research strategies because measurements can not reflect subtle tool interactions with the Testbed user.

COST:

5-year objectives: 2 man years. 10-year objectives: 3 man years for adopting estimation capabilities developed under other efforts (see "Contingencies"), 1 man year to provide user-preference measurement.

CONTINGENCIES:

Dependent on: (1) Testbed integration effort: instrumentation of testbed requires tracking all testbed activities. (2) "Requirements evaluation" effort and "Estimation of cost, risk, time in system development; Performance and Execution costs Analysis" effort: measuring the quality of the requirements before/after tool or technique use.

CONCLUSION:

Testbed effectiveness is mostly a development issue, and might be incorporated in the "Testbed integration" effort. It is not clear whether research strategies such as estimators will be effective. Most of the relevant research falls under the two research issues referenced under "Contingencies".

AUTHOR NAME:

Stephen Sherman, Michael Konrad.

RET R&D Effort

ISSUE NAME: Testbed Effectiveness

OBJECTIVE:

Provide capabilities to:
(1) Measure user time/effort and testbed resources spent.
(2) Perform cost, risk, schedule estimation.
(3) Compare requirements quality before/after application of a tool or technique.

SCOPE:

Technical areas to be addressed: Testbed instrumentation. Cost, risk, schedule estimation.

TECHNICAL APPROACH:

The technical approach is more development than research.

Objective 1: Instrument Testbed project tracking functionality to note for each project each usage of a tool, collecting effort data and before/after requirements versions.

Objective 2: Utilize product metrics developed under the two research efforts referenced under "Background" for measurement of requirements at key times in a project, for determination of tool and technique effectiveness. Adopt as appropriate to measurement of prototypes.

Objective 3: Ensure functionality exists to highlight differences between two versions of requirements, one ancestral to the other.

BACKGROUND:

The issue is unique. There are no similar efforts. References on estimation will be found in the background descriptions of two other research issues: "Requirements evaluation" and "Estimation of cost, risk, time in system development; Performance and Execution costs Analysis".

DURATION:

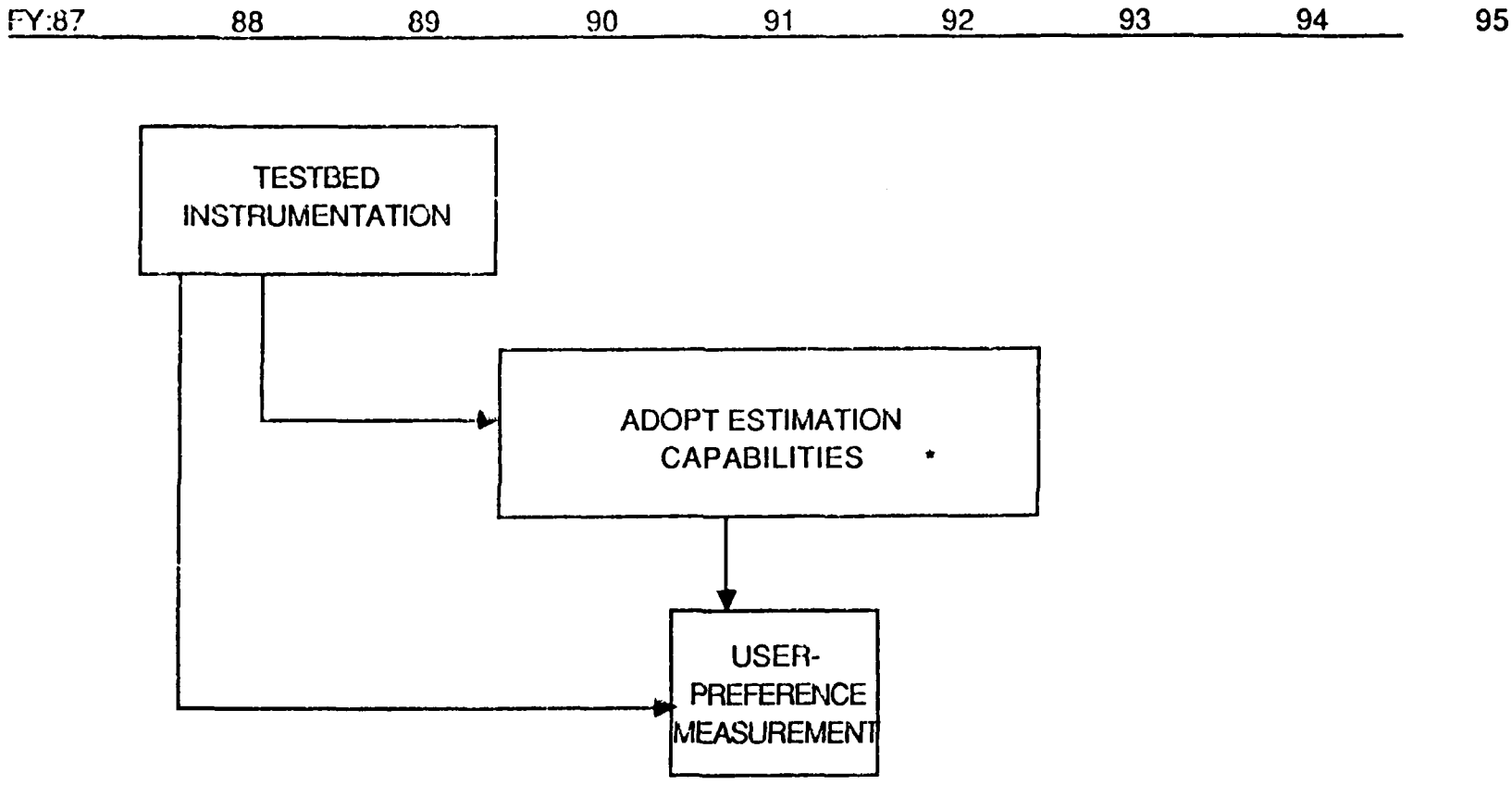
Testbed instrumentation	24 months
Adopt estimation capabilities	36 months
User-preference measurement support	12 months.

COST:

Testbed instrumentation	.3 million
Adopt estimation capabilities	.45 million
(assuming not covered by another effort)	
User-preference measurement support	.15 million.

DELIVERABLES:

PRODUCT:	DATE (months after start):
Testbed Instrumentation:	
Design (project tracking)	12 months
Code	24 months
Adopt estimation capabilities	
Identification of metrics to be employed (report)	12 months
Estimating techniques on prototypes (report)	18 months
Metrics for requirements descriptions (code)	24 months
Metrics for prototype descriptions (code)	36 months
User-preference measurement support	
Report on what evaluations Testbed user should provide on project completion	12 months



* ESTIMATION CAPABILITIES ADOPTED FROM RESEARCH:
"REQUIREMENTS EVALUATION" AND
"ESTIMATION OF COST, RISK, TIME IN SYSTEM DEVELOPMENT, PERFORMANCE & EXECUTION COSTS ANALYSIS"

Figure E.1.12-1 TESTBED EFFECTIVENESS R&D EFFORT

E.1.13 User Interface

RET Development Issue. Evolutionary Track.

PROBLEM DESCRIPTION:

The user interface is really the "packaging" associated with all the other requirements technology activities. All prototype and operational tools being developed should pay particular attention to this issue. All tool developers shall be required to use a consistent approach to end-user communication.

VALUE IN REQUIREMENTS PROCESS:

Can make the difference between whether or not the new technology is used or not. Vitally important for technology transfer and infusion in the mainstream of projects.

SOLUTION APPROACH:

A consistent and evolutionary approach to the development of user interfaces must be developed and supported. In order to achieve this the following steps must be performed:

- A characterization must be developed for the different types of users of the requirements tool set.

- A consistent user interface model must be developed for presenting and dealing with menus, as well as textual, graphical, and forms notations. Priority shall be placed on following a Xerox Parc - Apple Mac - Sun - Apollo type interface.

- Run time support capabilities must be specified and procured (or developed, if necessary) for supporting:

  * Virtual device interfaces for all I/O devices (e. g. graphics/textual screens of various resolution and color capabilities, printers and plotters of varying capabilities, keyboard and pointer devices of varying types).

  * Graphics support capabilities of at least GKS or higher level.

* Windowing support capabilities allowing the user to easily manipulate multiple concurrent tasks and manage multiple views of independent data bases/files.

* User-input dialog support capabilities for uniformly communicating with the end user in a consistent and friendly manner.

* Clipboard support capabilities for communicating arbitrary textual and graphical information between various tools/applications. (This capability should also allow and support tool developer's own definitions of complex data (self-defining data structures which can be passed between cooperative tools or tool fragments) - for example one might wish to pass a tree or symbol table of some sort between tools.)

5 YEAR OBJECTIVES:

Clearly establish user interface models and standards to be applied to all prototype and operational tool development activities.

10 YEAR OBJECTIVES:

Modify and maintain user interface models and standards in an evolutionary manner as appropriate.

RISK:

The only real risk associated with this activity is the risk of NOT doing it. If this is ignored or not attended to, there is a real high probability of much of the research effort going for naught.

COST:

There must be an initial effort aimed at specifying a set of candidate standards and conventions to be observed in the development of tool interfaces. This is probably on the order of a 1/2 to 1 person year activity. This initial effort will define a set of run time support libraries (or Packages) that should be used by subsequent tool builders.

The cost of the run time support capabilities will depend upon whether existing capabilities are available or new ones must be created. (For example, Windowing and graphics run time support will clearly be required - whether suitable capabilities are available or must be built will have to be determined.)

Subsequent costs should really be transparent to the main research activities and included in all tool development activities as standard fare.

CONTINGENCIES:

Since this is clearly a very central issue - it must be initiated early so subsequent tool development activities will be appropriately conducted.

CONCLUSION:

This is clearly a central and pervasive issue which affects all tool prototyping and development. It should be initiated very early in the program schedule and will have an ongoing effect on all subsequent activities.

In actuality it is really the "packaging" portion for all of the technology being developed. As such, it will make or break the transfer of technology to the use. Regardless of how tempting it might be to "ignore the packaging" initially, IT CAN NOT BE IGNORED.

AUTHOR NAME:

Leon G. Stucki, Michael Konrad (coauthor of "RET R&D effort")

RET R&D Effort

ISSUE NAME: User Interface

OBJECTIVE:

(1) Develop standards and conventions to be observed in the development of tool interfaces and the use of run-time support packages.

(2) Check compliance with standards by all tool contractors.

(3) Modify standards in an evolutionary manner.

SCOPE:

Broad standards are needed early. With time, selected standards will be revised and refined. Compliance checking should be thorough.

TECHNICAL APPROACH:

Objective 1 is already well addressed in the "Solution approach" and "Cost" sections. Objective 2 might be accomplished through a Quality Assurance function (perhaps associated with testbed administration) at RADC. Objective 3 might be accomplished through periodic reviews of standards for possible modification.

An alternative approach to satisfying objectives 2 and 3 is to appoint an independent authority who is responsible for both checking compliance and maintainirg and evolving standards. This is the approach taken below.

BACKGROUND:

Interactive user interfaces stress current underlying system facilities for input/output [1]. However, there are notable trends as evidenced recently in: (1) "Macintosh Style Manual" standards and (2) Windows™ tool interface and run-time support standards.

REFERENCES:

[1] M. Shaw, "An Input-Output Model for Interactive Systems", CHI'86 Proceedings, April 1986, ACM.

DURATION:

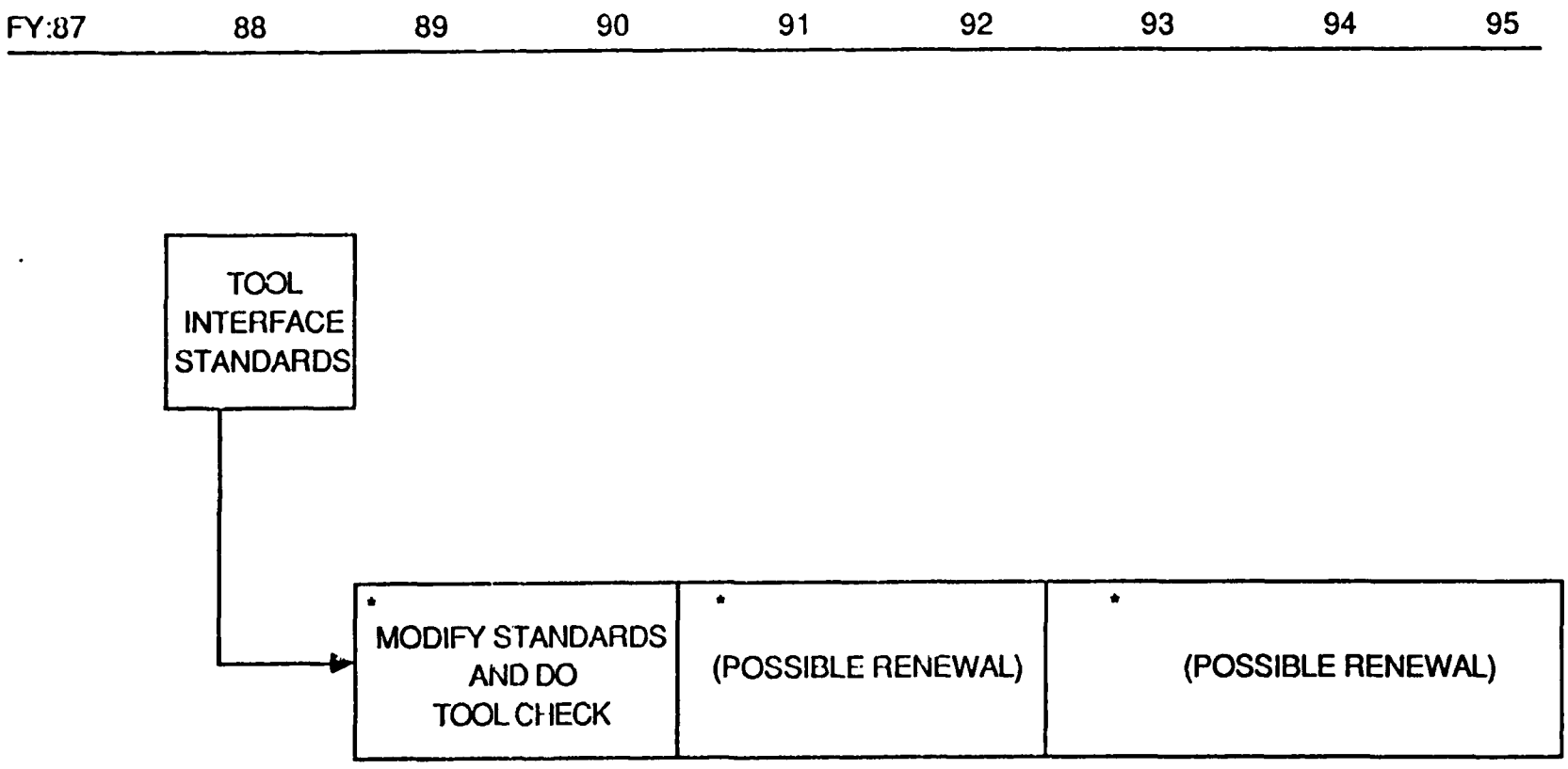
Tool interface standards	12 months
Modify standards and do tool check	24 months
(possible renewal every 24 months)	

COST:

Tool interface standards (cost assumes standard run-time packages do not have to be developed).	.15 million
Modify standards and do tool check	.15 million each 24 months

DELIVERABLES:

PRODUCT:	DATE (months after start):
Tool interface standards (3 reports:)	
User characterizations	3 months
User interface model for menus, text, graphics, forms	6 months
Standards (on tool interface and run-time support packages).	12 months
Modify standards and do tool check:	
Report on tool compliance with standards	every new tool
Revised standards	every 12 months



* ASSUMES EXISTING QUALITY ASSURANCE CANNOT HANDLE; POSSIBLE RENEWAL EVERY 24 MONTHS

Figure E.1.13-1 USER INTERFACE R&D EFFORT

E.1.14 Database

RET Development Issue. Evolutionary Track.

Database - Shared Data

PROBLEM DESCRIPTION:

Provide data storage and data management capabilities to support tight integration of RET tools via share data.

VALUE IN REQUIREMENTS PROCESS:

Sharing of data would occur on the following objects: data descriptions, scenario data, requirements descriptions, procedure descriptions, domain information, other text. The value of sharing comes in: minimizing reentry of common data; fast conveyance of outputs from one tool to inputs of other tools; assurance of data consistency between tools; maximizing the availability of data for each tool; and minimizing development of data interfacing code.

SOLUTION APPROACH:

Purchase a commercial database system which:

1) efficiently supports the data objects used in software design activities; and

2) provides standard levels of support for distributed data access, security, reliability, etc., for large volumes of data. While no such DBMS now exists, as one becomes available it should be incorporated in the RET. Also needed will be the definition of data descriptions for standard data objects used in communication between tools. The assumed solution approach is to have common data stored in a canonical form in the database and use input/output conversion facilities to put external data into the format required by each tool.

5 YEAR OBJECTIVES:

Within five years, a fully effective DBMS should be in place, and conversion standards and/or techniques for most tools will be in use.

10 YEAR OBJECTIVES:

Incorporate further advances in DBMS technology into the RET, as appropriate.

RISK:

The chance of obtaining a good DBMS is high, but not certain.

COST:

1) Evaluate, obtain, and install a DBMS;

2) develop common data object descriptions; and

3) Create conversion routines for RET tools in existence.

About one to three man-years.

DEPENDENCIES:

None within RET

CONCLUSION:

This is a critical element of the evolutionary RET track.

AUTHOR NAME:

Terry Welch

RET Development Issue. Evolutionary Track.

Database - Data Management Facilities

PROBLEM DESCRIPTION:

Facilities are needed to manage complex design information in the requirements development environment: data selection, report generation, and storage of relationships. This would apply to all design data: requirements text, prototype specifications, data descriptions, project management data, etc. Critical facilities would support managing structured text as a complex of design objects, and support viewing of data structures from a variety of projections (e.g. selecting a subset of a large system description and reordering that slice of the description according to some new criterion).

VALUE IN REQUIREMENTS PROCESS:

These facilities will be necessary to

1) extract particular subsets of information for specific jobs, such as preparing reports for specialized audiences;

2) analyze prior project information for reuseability; and

3) trace decisions between various parts of the requirements design process.

SOLUTION APPROACH:

Use a general-purpose DBMS which is efficient in the storage of design objects, and which provides the proper facilities.

5 YEAR OBJECTIVES:

Such a system should be in place within 5 years.

10 YEAR OBJECTIVES:

Incorporate further advances in DBMS technology into the RET, as appropriate.

RISK:

Some kind of system can always be found. The risk is that full capabilities for dealing with interrelated structures will not be found.

COST:

Evaluate, select, install, customize for RET needs: 1 man-year.

DEPENDENCIES:

This DBMS should be the same one used for sharing data between RET tools. The sharing function will more strongly stress the data modeling capabilities of the DBMS.

CONCLUSION:

This will be the highest pay-back, lowest cost element in the RET. It must, however, await the development of a proper DBMS for data sharing, so that the database will be populated with relevant project information.

AUTHOR NAME:

Terry Welch

RET R&D Effort

ISSUE NAME: Database - Shared Data & Data Management Facilities

OBJECTIVE:

Provide facilities which aid RET users and RET tools in collecting, accessing and managing large volumes of complex data, including

1) Viewing/Reporting - the ability to extract specified subsets of data elements and present those subsets of data in a way meaningful within application areas, often employing graphical presentation means.

2) Editing - Directly collecting and modifying database information.

3) Classification - capture of information from specific applications, and organization of that information to make it meaningful for later access.

4) Export/Import - formatting data for use by external tools.

SCOPE:

It is expected that this effort will be based on use of an existing commercial database system which has these properties:

1) it efficiently stores design representations, meaning complex object structures;

2) it provides conventional viewing and report generation facilities; and

3) support for team development through version control and distributed access. The proposed effort for the RET occurs in defining effective application specific data models and classifications, so that the general tools of DBMS are available through the RET standard user interface and through procedural interfaces.

TECHNICAL APPROACH:

This work involves conventional systems analysis for a non-conventional application, namely understanding application data types and information flow, and mapping those onto the tools provided by the DBMS. A critical aspect of the work will be the choice of the DBMS; many possible candidates will be poorly matched to this job.

BACKGROUND:

Many of the conceptual issues being addressed in this effort are also being studied elsewhere:

1) the Air Force directed VHSIC effort on Engineering Information Systems (EIS); and

2) efforts by several small companies to build object-centered design database systems. The concepts involved in the storage and access of design information should mature significantly over the next couple of years.

REFERENCES:

Not yet publicly available.

DURATION:

Once a suitable DBMS is available, an eighteen-month effort should suffice.

COST:

.45 million assuming a 3 man-year effort.

DELIVERABLES:

Software to be executed on the RET (Apollo system plus licenses for incorporated DBMS and graphics software packages as appropriate). Reports on analysis of user data and procedures should be available in six months, a DBMS selected and installed in twelve months, and application/tool specific interfaces provided in eighteen months.

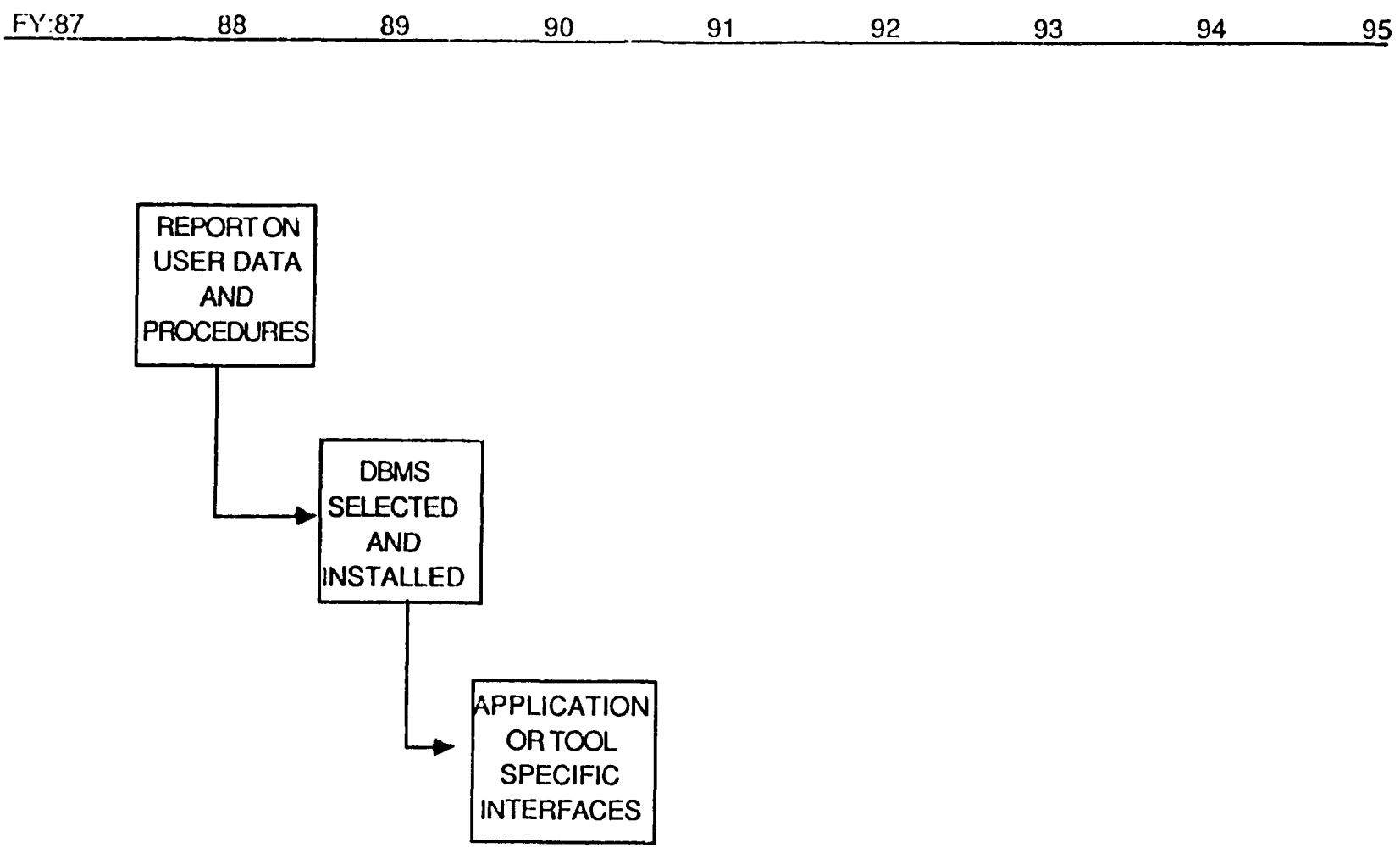


Figure E.1.14-1 DATABASE R&D EFFORT

E.1.15 Evolutionary Testbed Integration

RET Development Issue. Evolutionary Track.

PROBLEM DESCRIPTION:

To maximize the benefit from the testbed, all developed or acquired tools should be capable of working together, and thus they need to be integrated. Also, a key purpose of the testbed is to support experiments comparing different tools and techniques; this requires monitoring tool use and controlling parameters. Such is made easier in an integrated testbed.

VALUE IN REQUIREMENTS PROCESS:

Key capabilities of different tools can be brought to bear on the same requirements problems, leading to improved requirements.

SOLUTION APPROACH:

To "bring" the tools together: modify tools to utilize the common database and common user interface developed under "Database" and "User Interface" development issues. Instrument the testbed for tool tracking and measurement.

To make effective use of the testbed, a "Testbed usage methodology" should be developed.

5-YEAR OBJECTIVES:

An integrated testbed:
tools have a common user interface and database,
testbed activities tracked and measured, and a
testbed usage methodology.

10-YEAR OBJECTIVES:

Incorporate new tools and techniques.
Further testbed evolution toward data-orientation from tool-oriented interface.

RISK:

One risk is integration made unnecessarily tight (wasted effort). Determining when functionality is sufficiently mature for data-

directed invocation is tricky. Current RADC contractors need to cooperatively develop their tools to aid later integration.

COST:

5-year objectives: 3.5 man years. 10-year objectives: 3 man years.

CONTINGENCIES:

This development issue should subsume the testbed instrumentation activity discussed in "Testbed effectiveness". Providing a common user interface and database for the tools and standards is addressed under "User Interface" and "Database" development issues. The capability for data-directed invocation should be covered by the "Database" issue.

CONCLUSION:

To make the testbed greater than the sum of individual tools, the Evolutionary testbed integration effort must be funded. The case for funding long-term objectives is less clear.

AUTHOR NAME:

Stephen Sherman, Michael Konrad.

RET R&D Effort

ISSUE NAME: Evolutionary Testbed Integration

OBJECTIVE:

Develop an integrated testbed featuring:

(1) a common database and user interface for tools,
(2) capability for evolution toward data-directed invocation,
(3) tracking and measurement of testbed activities,
(4) a testbed usage methodology, and
(5) a way of bringing other tools in.

SCOPE:

Technical areas to be addressed: Experiment Methodologies.

TECHNICAL APPROACH:

First determine the minimal degree of integration necessary to get the tool interaction and tracking desired. Identify what functionality should be automatically invoked by changes in data, and what should be under explicit user control. Identify what aspects of tool use are to be tracked; modified tools should automatically record details of each usage. In short, all user and tool activities will be registered in the common database, providing a basis for project tracking and measurement.

A testbed usage methodology should guide experimental use of the testbed facility. Each user's project (experiment) should include a pre-testbed phase identifying experiment hypothesis and a post-testbed analysis of results.

Future contractors should develop their tools to directly fit in the testbed (use common database, accessed through common interface).

A subsequent integration contract will address evolution of the testbed: getting new functionality to work with the old, getting more functionality automatically invoked, and improving what is tracked.

BACKGROUND:

The issue is unique. There are no known similar efforts. The references cite the currently planned prototyping tools that will first be integrated.

REFERENCES:

[1] M. Stephens and K. Whitehead, "The Analyst - A Workstation for Analysis and Design", Proc. 8th Int. Conf. Software Eng., IEEE Comp. Soc. Press, 1985.

[2] R. Yeh and M. Konrad, "VHLL System Prototyping Tools", a proposal submitted to Rome Air Development Center.

[3] P. Daley, "C^3I Rapid Prototype Investigation", Rome Air Development Center, RADC-TR-85-216.

DURATION:

Integration of current tools	24 months
Testbed usage methodology	12 months
Integration of future tools	36 months

COST:

Integration of current tools	.45 million
Testbed usage methodology	.07 million
Integration of future tools	.45 million

DELIVERABLES:

PRODUCT:	DATE (months after start):
Integration of current tools	
Report on degree of integration required	5 months
Modified tools (design)	9 months
Modified tools (code)	12 months
Testbed usage methodology (report)	12 months
Integration of future tools	
Identify functionality to be incorporated (report)	18 months
Modify tools (design)	28 months
Modify tools (code)	36 months.

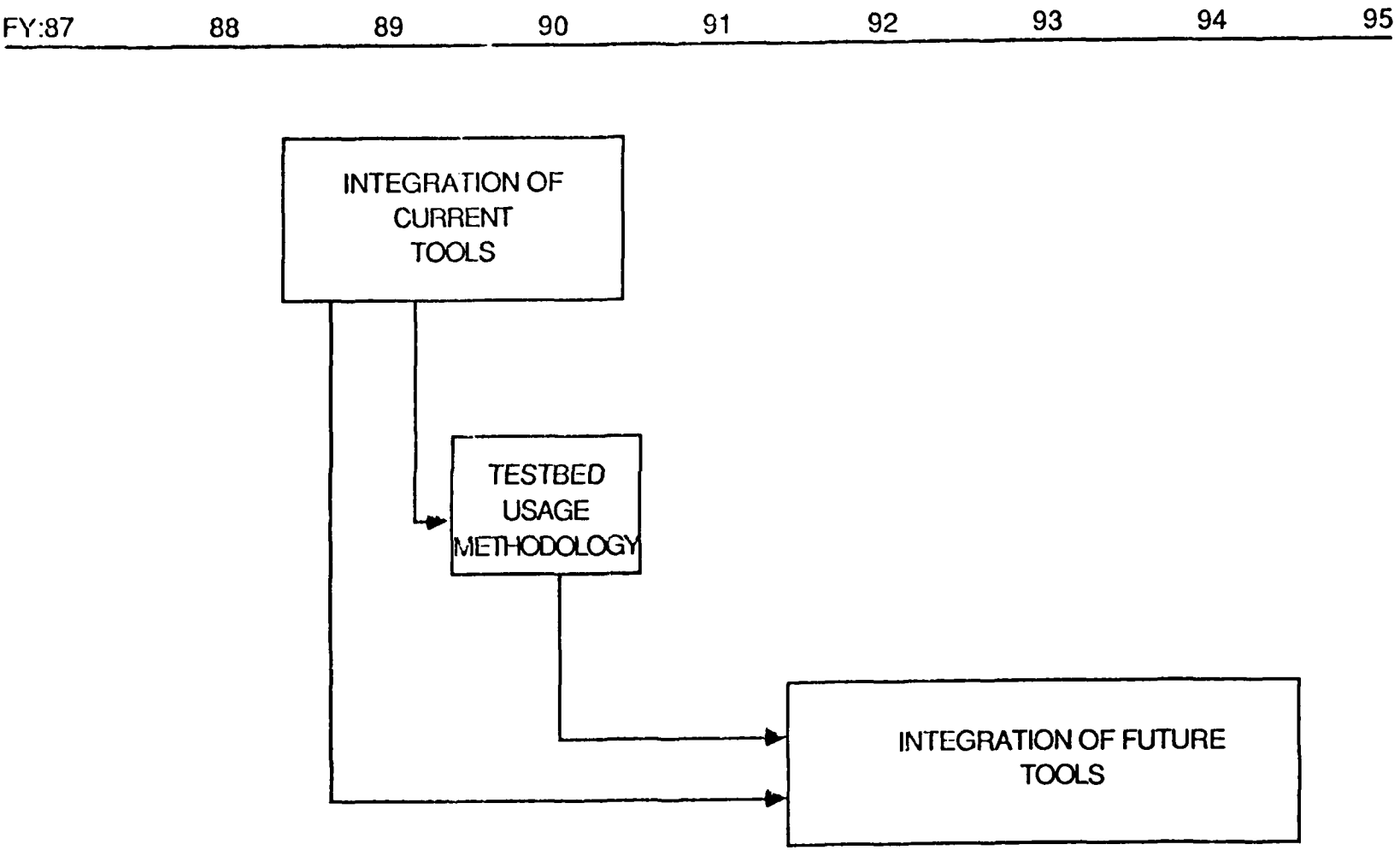


Figure E.1.15-1 INFORMAL TESTBED INTEGRATION R&D EFFORT

E.2 Formal Language Track

RET Research Issue. Formal Language Track.

PROBLEM DESCRIPTION:

Requirements are currently informal. Hence computer tools can do no more than keep track of the requirements statements and human claims about tracking and satisfaction (i.e., electronic note pad and record keeping). Requirements are thus merely guiding comments for human consumption and the requirements process is supported by management of people through methodology.

The goal is the replacement of this informal basis by a formal treatment of requirements and automated tool support for requirements design and tracking into a specification.

VALUE IN REQUIREMENTS PROCESS:

Extremely high. Value arises from earlier detection of inconsistent or unsatisfiable requirements, better trade-off analysis, and earlier detection of requirements unsatisfied in a specification, particularly ones only partially (or sometimes) satisfied.

SOLUTION APPROACH:

Expand existing formal specification language to include formal requirements statements. Share a common domain model and define requirements as predicates against behavior of specification. Formally execute specification to generate behavior against which to test requirements predicates. Include goals as requirements which are only "desired." Provide support for multiple levels of abstraction in stating requirements and specifications and mapping between them. Provide support for evolving requirements statement on basis of feedback from evaluation tools.

5 YEAR OBJECTIVES:

Common formal language for requirements, specifications, and goals which share the same domain and behavior models and methodologies for dealing with these formalisms. Provide tool which determines whether requirements are satisfied by a specification with respect to a particular scenario.

10 YEAR OBJECTIVES:

Expand requirements satisfaction determination tool to handle classes of scenarios (via symbolic execution) and automatic determination of scenarios. Support multiple levels of abstraction for requirements and specifications and the mappings

between them. Manage the human and computing resources engaged in the evolution of requirements.

RISK:

Relatively low for short term objectives to product integrated formal requirements and specification language and to formally execute a specification against a scenario to generate behavior, and to use that behavior to determine whether requirements are satisfied.

Relatively high over the long term to build reasoning and analysis tools which can handle practical size requirements statements and provide deep and comprehensive feedback to aid iterative design of requirements.

COST: 8.7 million

DEPENDENCIES:

Requires more highly trained Requirements Engineer and Mission User. Entire approach is dependent upon integration of requirements into Formal Specification language, but this is relatively low risk. Incremental requirements component assumes successful prior completion of corresponding incremental specification effort (separately funded).

CONCLUSION:

Formal language approach to requirements is highly recommended as complement to evolutionary track. It is higher risk and higher payoff and that payoff occurs later than in evolutionary track. But it lays the foundation for earlier and more reliable detection of requirements problems and their use as a real design envelope.

AUTHOR NAME:

Robert Balzer

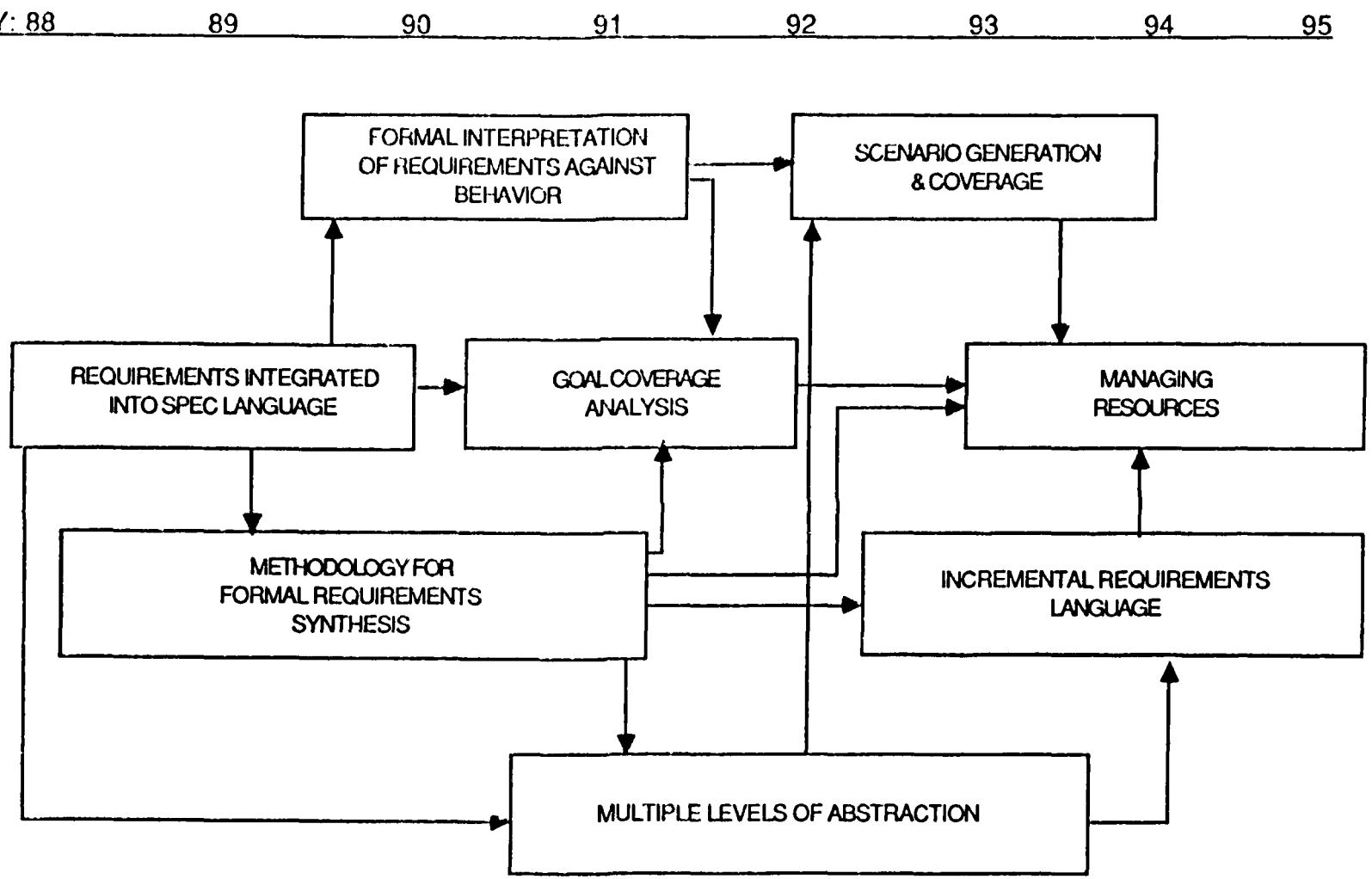


Figure E.2-1 FORMAL LANGUAGE APPROACH ROADMAP

E.2.1 Requirements Integrated into Spec Language

RET R&D Effort. Formal Language Track

OBJECTIVE:

Integrate requirements into formal specification language, sharing common domain and behavior models. Resultant language must be executable.

SCOPE:

Technical areas to be addressed: Formal semantics, domain models, database schema, executable specifications, formal specification languages.

Technical areas not addressed: Advances in formal semantic theory, new formal languages, symbolic evaluation.

TECHNICAL APPROACH:

Survey existing executable specification languages. Choose one with an explicit domain model and formal model of behavior. Define requirements as predicates against behavior and integrate into language.

RELEVANT WORK:

OBJ (SRI), Paisley (AT&T), Gist (ISI), Clear (Edinburgh University).

DURATION: 24 months

COST: 1 million

DELIVERABLES:

Formal requirements and spec language, shared domain model	12 months
Formal requirements and spec language, shared behavior model	24 months

E.2.2 Formal Interpretation of Requirements Against Behavior

RET R&D Effort. Formal Language Track

OBJECTIVE:

Execute specification against scenario to generate its behavior. Determine whether requirements were satisfied by this behavior.

SCOPE:

Technical areas to be addressed: Symbolic evaluation, formal semantics, temporal logic, executable specification.

Technical areas not addressed: Advances in formal semantic theory or logic.

TECHNICAL APPROACH:

Use symbolic evaluator to generate the temporal behavior of an executable specification on a scenario. Extend symbolic evaluator with additional phase that determines whether requirements are satisfied by this behavior. If not, inform user which requirements were violated by what portions of the behavior and how this resulted from the specification.

RELEVANT WORK:

Gist symbolic evaluator and behavior explainer (ISI), EL1 symbolic evaluator (Harvard), Paisley evaluator (AT&T), scenario specification.

DURATION: 24 months

COST: 1 million

DELIVERABLES:

Symbolic evaluator for requirements and spec language	12 months
Requirements predicate checker	21 months
Requirements violation checker	24 months

E.2.3 Methodology for Formal Requirements Synthesis

RET R&D Effort. Formal Language Track

OBJECTIVE:

Provide guidance for Mission Users and Requirements Engineers in creating formal requirements and using facilities of RET formal track.

SCOPE:

Technical areas to be addressed: None.

TECHNICAL APPROACH:

Extend structured specification methodologies to requirements and their formal expression. Define methodology for using requirements checker and modifying requirements statement. Define methodology for stating goals and determining which should become requirements. Design experiments to test and validate proposed methodologies.

RELEVANT WORK:

Structured Design, CORE, metrics.

DURATION: 36 months

COST: 1 million

DELIVERABLES:

Structured requirements synthesis methodology	12 months
Guidelines for use of requirements checker	24 months
Methodology for using goals	30 months
Requirements methodology manual	36 months

E.2.4 Goal Coverage Analysis

RET R&D Effort. Formal Language Track

OBJECTIVE:

Provide feedback to user about degree of coverage of goal satisfaction and degree to which satisfaction is obtained. Provide basis for trade-off analysis.

SCOPE:

Technical areas to be addressed: Symbolic evaluation, test coverage analysis, decision support systems.

TECHNICAL APPROACH:

Extend requirements checker to keep track of cases for which goals were not satisfied. Characterize degree of coverage from model of search space. Define measures of satisfiability and algebra for combining values.

RELEVANT WORK:

DURATION: 24 months

COST: 1 million

DELIVERABLES:

Tracker of unsatisfied goals	6 months
Model of search space and estimator of degree of goal coverage	12 months
Measures of satisfiability and algebra for their combination	24 months

E.2.5 Multiple Levels of Abstraction

RET R&D Effort. Formal Language Track

OBJECTIVE:

Formalize tracking of requirements through multiple levels of abstraction.

Requirements are predicate against behavior. Hence, refinement results in one or more requirements which together should imply the original requirement. The methodology for formal requirements synthesis will provide guidance for when and how such refinements occur. This effort will ensure the validity of each such refinement, keep track of which higher level requirement(s) are (partially) satisfied by a lower level one, and identify what extra assumptions/commitments resulted from the refinement.

SCOPE:

Technical areas to be addressed: Formal models, abstract data types.

Technical areas not addressed: Automatic classification, knowledge representation theory.

TECHNICAL APPROACH:

Provide formal basis for defining abstract models and for relating activity in one to the corresponding activity in the other. Provide tool which verifies that a requirement in one model is ensured by a set of requirements in another model. Provide a tool which finds that set, if it exists (this is a formal requirements tracker).

RELEVANT WORK:

Automatic classification, knowledge representation, theorem proving.

DURATION: 36 months

COST: 1.5 million

DELIVERABLES:

Abstract Model Definition facility	12 months
Verifier that a requirement in one model is covered by a set of refinements in another model	24 months

Identification of extra assumptions/commitments resulting from a refinement	30 months
Formal Requirements Tracker	36 months

E.2.6 Scenario Generation and Coverage

RET R&D Effort. Formal Language Track

OBJECTIVE:

Scenarios are crucial to testing requirements and the systems that (purport to) implement them. Whereas requirements and specifications are general statements, which are, respectively, predicates against behavior and generators of that behavior, scenarios define behavior for specific cases. As such, they can be used as test cases to verify that both requirements and specifications include the desired specific behavior.

However, to be used in this manner, a scenario must be both complete and detailed. It must not only define the desired behavior, but also all the inputs and controls needed to ensure that this behavior will result. Furthermore, some comparison mechanism must exist to determine whether the behavior generated by the specification, which is more complete and detailed than the subset specified in the scenario, is a valid instantiation of that desired behavior.

These requirements currently make scenario generation a demanding, labor intensive, error-prone, and time-consuming task. However, with formal requirements and specifications, much of this effort can be automated.

The goal of this effort is to generate a complete scenario from a mission user's outline of desired behavior.

SCOPE:

Technical areas to be addressed: Test case generation, symbolic evaluation.

TECHNICAL APPROACH:

Reverse the flow of reasoning in symbolic evaluation so that, for a given specification, partially specified behavior (i.e., symbolic output) can be used to characterize the inputs necessary to generate that behavior. Such a "backward evaluation" would build up parameterized expressions and constraints which defined the sets of inputs that would generate the behavior. Consistently instantiating these expressions (without violating the constraints) would provide a specific scenario for the desired behavior.

Combinatorial explosion in the search space will necessitate the use of heuristics to control both the "backward evaluation" and the parameter instantiation processes. These heuristics should

combine knowledge of generating test cases and of normal situations and behaviors in the target domain.

Capabilities produced for mapping and matching multiple levels of abstraction would be used to determine whether a specific behavior generated from a specification or prototype satisfied the mission user's statement of desired behavior.

RELEVANT WORK:

Symbolic Evaluation, Test case generation, multiple levels of abstraction, heuristic search, diagnostic systems.

DURATION: 24 months

COST: 1 million

DELIVERABLES:

Scenarios generation from Mission User's behavior spec	18 months
Determination of satisfaction of actual behavior against Mission User's behavior spec	24 months

E.2.7 Incremental Requirements Language

RET R&D Effort. Formal Language Track

OBJECTIVE:

Provide language-based support for evolving requirements rather than creating them from scratch all at once.

Requirements evolve as Mission Users, Acquisition Engineers, and Developers think more carefully about the system to be built and get feedback, insight, and experience, from analysis tools and/or prototypes. Yet, formal languages capture none of this time history. Each stage is a stand-alone complete description related to the others only by a version number. Each stage must be produced by low-level text edits of the previous stage which effect the intended modification but keep that modification implicit.

This effort directly supports evolution by providing explicit language constructs for common modifications. These constructs increase comprehension for both humans and tools by focusing attention on the changes and by providing an incremental, staged basis for understanding.

SCOPE:

Technical areas to be addressed: Formal languages, transformations.

TECHNICAL APPROACH:

Build upon prior work in incremental specification languages for base of modification constructs, human reading tools, and tool focusing capabilities. Add special support for refining predicates (requirements), strengthening and weakening them, and revising them.

RELEVANT WORK:

Monotonic logics and languages, cognitive models, transformation, multi-level models.

DURATION: 36 months

COST: 1.5 million

DELIVERABLES:

Incremental requirements refinement constructs — 6 months

Incremental requirements strengthening and weakening constructs	12 months
Incremental requirements revision constructs	18 months
Integrate incremental language constructs with multi-level requirements tracking tool	36 months

E.2.8 Managing Resources

RET R&D Effort. Formal Language Track

OBJECTIVE:

Manage the human and computing resources needed to create an initial set of requirements, analyze them, gather feedback from prototypes (specifications), determine whether they are satisfied by the prototype's behavior, iteratively revise and refine them, and track them throughout this process.

SCOPE:

Technical areas to be addressed: Task representation, agenda management.

TECHNICAL APPROACH:

Provide formal support for activities informally defined by previous methodology guidelines effort. Formally represent the tasks required. Identify their preconditions, resources, and results. Construct manager which understands these dependencies and guides users in task selection.

RELEVANT WORK:

KBSA Activity Coordinator, CAD

DURATION: 24 months

COST: .75 million

DELIVERABLES:

Formal representation of RET tasks	6 months
Task manager for single user	12 months
Task and tool manager for single user	18 months
Task and tool coordinator for multi-user requirements effort	24 months

APPENDIX F: PHILOSOPHICAL CONSIDERATIONS

F.1 Assessing Completeness in Requirements

Determining whether requirements are complete in the general case is impossible. However by carefully limiting oneself to a finite number of well-defined properties, and defining completeness as satisfaction of these properties, it is possible to determine whether the requirements are complete. An example set of properties (from [1]) is:

* No TBDs,
* No nonexistent references,
* No missing specification items,
* No missing functions,
* No missing products.

The problem in the general case is that we don't know all the properties because we lack good meta-models covering all that which we are trying to specify.

References:

[1] B. Boehm, "Verifying and Validating Software Requirements and Design Specifications", IEEE Software, January 1984.

F.2 Unexpected Bounding of the Solution

The mechanisms used to express requirements affect the choice of a solution. There is no resolution to this problem.

Simply stating a problem necessarily bounds the corresponding solution space. Formal expression exacerbates the bounding (there is less ambiguity). Even the formalization of an application domain which is subsequently referenced in several requirements imposes bounds on the corresponding solution spaces.

Where does the bounding come from? Formal descriptions are necessarily a mixture of organizational/structural expressions (designs) and expressions of bounds. While the describer's goal is the expression of the latter (the bounds), their expression must be made in the context of the former (the designs). The problem is that these designs themselves produce bounds, and these are in addition to those bounds the describer may have intended to explicitly state.

The conclusion of this is that requirements and solution architectures contain similar kinds of descriptions (designs and bounds) and so there should be a sharing of the syntax and semantics used in expressing them.

Testbed users need to be made aware that the form of expression they employ may bound the solution space in undesirable ways.

APPENDIX G: REFERENCES

Alavi, M., "An Assessment of the Prototyping Approach to Information Systems Development", Communications of the ACM, Vol. 27, No. 6, June 1984.

Alford, M., "A Requirements Engineering Methodology for Real Time Processing Requirements", IEEE T.S.E., January 1977.

Arrango G., Freeman P., "Modeling knowledge for software development" Third International Workshop on Software Specification and Design 1985, IEEE Computer Society.

Barstow, D., "Domain-Specific Automatic Programming", IEEE T.S.E., November 1985.

Bartlett A.J., Cherrie B.H., Lehman M.M., MacLean R.I., and Potts C., "The Role of Executable Metric Models in the Programming Process", Final Report, ARPA Order No. B-2-3288.

Basili V. and Weiss D.M., "A Methodology for collecting valid Software Engineering data", IEEE T.S.E., Vol. SE-10, November 1984.

Beck L.L., Perkins T.E., "A Survey of Software Engineering Practice: Tools, Methods and Results", IEEE T.S.E. SE-9, September 1983.

Bell et al, "An Extendable Approach to Computer-Aided Software Requirement Engineering", IEEE T.S.E., SE-3(1), January 1977.

Bell T.E., Thayer T.A., "Software Requirements: Are they really a Problem", Proceeding 2nd International Conference on Software Engineering, 1976 IEEE.

Boehm B.W., Software Engineering Economics, Prentice Hall 1981.

Boehm B.W., "A Spiral Model of Software Development and Enhancement," 1985, TRW tech. report 21-371-85, TRW, Inc., 1 Space Park, Redondo Beach, CA 90278.

Boehm B.W., Brown J.R., Lipow M., "Characteristics of Software Quality", New York: North Holland, 1978.

Buchanan B.G., Feigenbaum E.A., "DENDRAL and Meta-DENDRAL", Artificial Intelligence 11:1 (1978) pp.5-24.

Clark K.L., McCabe F.G., "PROLOG: a language for implementing Expert Systems", Machine Intelligence.

Daley, P., "C^3I Rapid Prototype Investigation", Rome Air Development Center, RADC-TR-85-216.

Davis, A., "The Design of a Family of Application-Oriented Requirements Languages", IEEE Computer, May 1982.

Ehrig H., Mahr B., Fundamentals of Algebraic Specification 1: Equations and Initial Semantics, EATCS Monographs on Theoretical Computer Science, Vol. 6, Springer-Verlag, New York, 1985.

Finkelstein, A. and Potts, C., "Structured Common Sense: The Elicitation and Formalization of System Requirements", Research Report #86/7, Imperial College, 180 Queens Gate, London, SW7 2BZ.

Foggy, C., "The OPS-5 User's Manual" Technical Report, Carnegie Mellon University 1980.

Friedman F.R., et al, "Price Software Model - Overview", Internal paper, Price system, RCA Corporation, Cherry Hill, NJ, 1979.

Greenspan S. "Requirement Modeling: A Knowledge Representation Approach to Software Requirements Definition" CSRG, U. of Toronto, Tech. Rep. CSRG-155, March 1984.

Greenspan, S. J., Mylopoulos, J., and Borgida, A., "Capturing More World Knowledge in the Requirements Specification", Proc. 6th Int. Conf. Software Eng.

Guttag J., Horning J., "Formal Specification as a Design Tool", Proc. ACM symp. Princ. of Programming Languages, January 1980, pp. 251-261.

Heninger K.L., "Specifying Software Requirement for Computer System: New Techniques and their Applications", IEEE T.S.E., SE-6(1), January 1980.

Henry S., Kafura D., Harris K., "On the relationship among three software metrics", Performance Evaluation Review, Vol. 10 Number 1, 1981 pp. 3-10.

Konrad M., Welch T., "VHLL System Prototyping Tool - Users Manual", Int'l Software Systems, Inc., Austin, TX 78759, (1987).

Leung C.H.C., Choo Q.H. "A Knowledge-base for effective Software Specification and Maintenance", Third International Workshop on Software Specification and Design 1985, IEEE Computer Society.

McArthur, D., Klahr, P., Narain, S., "ROSS: An Object-Oriented Language to Constructing Simulations", R-3160-Af, December 1984.

McCabe T.J., Young L.F. et al, "Design Basis Paths: A Complexity Driven Design Inspection Methodology", Proceedings of Total System Reliability Symposium, Gaithersburg, MD, December 1983.

McCall J.A., et al. "Factors In Software Quality", Tech. rep. 77c1s62 Sunnyvale, CA: General Electric; Command and Information Systems, 1977.

Miyamoto, Yeh R.T., "A Software Requirement and Definition Methodology for Business Data Processing", Proc. 1981 NCC AFIPS Press 1981.

Nau, D., "Expert Computer Systems", Computer Feb. 1983, pp. 63-85.

Neighbors, J., "The Draco approach to constructing software from reusable components", IEEE T.S.E., vol SE-10, September 1984.

Ramamoorthy C.V., Garg V. and Aggarwal R. "Environment Modeling and Activity Management in GENESIS", 2nd conference on software tools, techniques and development 1985.

Ramamoorthy C.V., So S.S. "Software Requirements and Specifications: Status and Perspectives", draft report from University of California at Berkeley.

Ramamoorthy C.V., Tsai W.T., Yamaura T. and Bhide A., "Metric Guided Methodology", COMPSAC 1985.

Ramamoorthy C.V., Usuda Y., Tsai W. and Prakash A., "GENESIS: An Integrated Environment for supporting Development and Evolution of Software", COMPSAC 1985.

Reddy, Y., Fox, M., "KBS: An Artificial Intelligence Approach to Flexible Simulation", CUM-RI-TR-82-1, Robotics Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 15213, 14 September 192.

Schoman and Ross, "Structured Analysis for Requirement Definition", IEEE T.S.E., SE-3(1), January 1977.

Shapiro, E., Algorithmic Program Debugging, The MIT Press, Cambridge, Massachusetts, London, England.

Shaw, M., "An Input-Output Model for Interactive Systems", CHI'86 Proceedings, April 1986, ACM.

SimKit is a commercial product of IntelliCorp.

SimulCraft is a commercial product of the Carnegie Group.

Stachowitz, Bar-Hillel, Winograd, Simmons, "Requirements for Mechanical Translation - Problems, Solutions, Prospects" in Feasibility Study on Fully Automatic High Quality Translation, Austin, Texas, Linguistics Research Center, The University of Texas at Austin, RADC-TR-71-295, 1971.

Stephens, M. and Whitehead, K., "The Analyst - A Workstation for Analysis and Design", Proc. 8th Int. Conf. Software Eng., IEEE Comp. Soc. Press, 1985.

Stone, A., Hartschuh, D., and Castor, B., "SREM Evaluation", Rome Air Development Center, RADC-TR-83-314.

Stucki, L., "New Directions in Automated Tools for Improving Software Quality", Current Trends in Programming Methodology, Vol. II, Yeh, R. T. editor, Prentice Hall, 1977.

Sundfor, S., "Draco Domain Analysis for a Real Time Application: The Analysis", Tech Report RTP 015, Irvine, CA: University of California, 1983a.

Sundfor, S., "Draco Domain Analysis for a Real Time Application: Discussion of the Results", Tech Report RTP 016, Irvine, CA: University of California, 1983b.

Teichrow, Hershey "PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Info Processing Systems.", IEEE T.S.E., SE-3(1), January 1977.

Vick C.R., Davis C.G., "The Software Development System", IEEE T.S.E., SE-3(1), January 1977.

Yeh, R. and Konrad, M., "VHLL System Prototyping Tools", a proposal submitted to Rome Air Development Center. See Konrad M., Welch T. reference.

Walston C.E., Felix C.L., "A Method of Programming Measurement and Estimation", IBM Systems Journal Vol. 16, No. 1, 1977.

Wasserman, A.I., "Extending State Transition Diagrams for the Specification of Human-Computer Interaction", IEEE T.S.E., August 1985.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.